

No. 23-2265

---

---

**United States Court of Appeals  
for the Federal Circuit**

---

---

EXPRESS MOBILE, INC.,

*Plaintiff-Appellant,*

v.

GODADDY.COM, LLC,

*Defendant-Appellee.*

---

Appeal from the United States District Court for the District of Delaware  
in C.A. No. 19-cv-01937, United States District Court Judge Matthew F. Kennelly

---

---

**PRINCIPAL BRIEF OF PLAINTIFF-APPELLANT**

---

James R. Nuttall  
(Lead Counsel)  
jnuttall@steptoe.com  
Katherine H. Tellez  
Robert F. Kappers  
Candice J. Kwark  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300  
Facsimile: (312) 577-1370

November 13, 2023

*Counsel for Plaintiff-Appellant,  
Express Mobile, Inc.*

**EXEMPLARY PATENT CLAIMS ILLUSTRATIVE OF THE ISSUES**

U.S. Patent No. 6,546,397

1. A method to allow users to produce Internet websites on and for computers having a browser and a virtual machine capable of generating displays, said method comprising:
  - (a) presenting a viewable menu having a user selectable panel of settings describing elements on a website, said panel of settings being presented through a browser on a computer adapted to accept one or more of said selectable settings in said panel as inputs therefrom, and where at least one of said user selectable settings in said panel corresponds to commands to said virtual machine;
  - (b) generating a display in accordance with one or more user selected settings substantially contemporaneously with the selection thereof;
  - (c) storing information representative of said one or more user selected settings in a database;
  - (d) generating a website at least in part by retrieving said information representative of said one or more user selected settings stored in said database; and
  - (e) building one or more web pages to generate said website from at least a portion of said database and at least one *run time file*
  - (f) where said at least one *run time file* utilizes information stored in said database to generate virtual machine commands for the display of at least a portion of said one or more web pages.

U.S. Patent No. 7,594,168

1. A system for assembling a web site comprising:

a server comprising a build engine configured to:

accept user input to create a web site, the web site comprising a plurality of web pages, each web page comprising a plurality of objects,

accept user input to associate a style with objects of the plurality of web pages,

wherein each web page comprises at least one button object or at least one image object,

and wherein the at least one button object or at least one image object is associated with a style that includes values defining transformations and time lines for the at least one button object or at least one image object;

and wherein each web page is defined entirely by each of the plurality of objects comprising that web page and the style associated with the object,

produce a database with a multidimensional array comprising the objects that comprise the web site including data defining, for each object, the object style, an object number, and an indication of the web page that each object is part of, and

provide the database to a server accessible to web browser; wherein the database is produced such that a web browser with access to a *runtime engine* is configured to generate the web-site from the objects and style data extracted from the provided database.

U.S. Patent No. 9,063,755

1. A system for generating code to provide content on a display of a device,  
said system comprising:

computer memory storing a registry of:

a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, and

b) the address of the web service;

an authoring tool configured to:

define a user interface (UI) object for presentation on the display, where said UI object corresponds to the web component included in said registry selected from the group consisting of an input of the web service and an output of the web service,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object,

produce an Application including the selected symbolic name of the defined UI object, where said Application is a device-independent code, and

produce a Player, where said Player is a device-dependent code;

such that, when the Application and Player are provided to the device and executed on the device, and when a user of the device provides one or more input values associated with an input symbolic name to an input of defined UI object,

1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,

2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,



3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for a display of the device to present an output value in the defined UI object.

U.S. Patent No. 9,471,287

1. A system for generating code to provide content on a display of a device, said system comprising

computer memory storing a registry of:

a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network,

where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service,

where each symbolic name has an associated data format class type corresponding to a subclass of User Interface (UI) objects that support the data

format type of the symbolic name, and has a preferred UI object, and

b) an address of the web service;

an authoring tool configured to:

define a (UI) object for presentation on the display, where said defined UI object corresponds to a web component included in said registry selected from a group consisting of an input of the web service and an output of the web service,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object,

where the selected symbolic name is only available to UI objects that support the defined data format associated with that symbolic name, and

produce an Application including the selected symbolic name of the defined UI object, where said Application is a device-independent code, and

produce a Player, where said Player is a device-dependent code;

such that, when the Application and Player are provided to the device and executed on the device, and when a user of the device provides one or more input values associated with an input symbolic name to an input of defined UI object,

- 1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,
- 2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,
- 3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for a display of the device to present an output value in the defined UI object.

U.S. Patent No. 9,928,044

1. A system for generating code to provide content on a display of a device, said system comprising:

computer memory storing:

a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic

names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service,

where each symbolic name has an associated data format class type corresponding to a subclass of User Interface (UI) objects that support the data format type of the symbolic name, and where each symbolic name has a preferred UI object, and

b) an address of the web service;

an authoring tool configured to:

define a UI object for presentation on the display, where said defined UI object corresponds to a web component included in said computer memory selected from a group consisting of an input of the web service and an output of the web service,

where each defined UI object is either: selected by a user of the authoring tool; or automatically selected by the system as the preferred UI object corresponding to the symbolic name of the web component selected by the user of the authoring tool,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object,

where the selected symbolic name is only available to UI objects that support the defined data format associated with that symbolic name,

store information representative of said defined UI object and related settings in a database;

retrieve said information representative of said one or more said UI object settings stored in said database; and

build an application consisting of one or more web page views from at least a portion of said database utilizing at least one player,

where said player utilizes information stored in said database to generate for the display of at least a portion of said one or more web pages,

wherein when the application and player are provided to the device and executed on the device, and when the user of the device provides one or more input values associated with an input symbolic name to an input of the defined UI object,

- 1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,
- 2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,
- 3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for a display of the device to present an output value in the defined UI object.

**CERTIFICATE OF INTEREST FOR**  
**EXPRESS MOBILE, INC.**

Pursuant to Federal Circuit Rules 28(a)(4) and 47.4, counsel for Plaintiff-Appellant Express Mobile, Inc. certifies the following:

1. The full name of every party represented by me is:

Express Mobile, Inc.

2. The name of the real party in interest (if the party named in the caption is not the real party in interest) represented by me is:

Not Applicable.

3. All parent corporations and any publically-held companies that own 10% or more of the stock of any party represented by me are:

Not Applicable.

4. The names of all law firms and the partners or associates that appeared for the party or amicus now represented by me in the trial court or agency or are expected to appear in this court (and who have not or will not enter an appearance in this case) are:

Michael Dockterman  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300

John L. Abramic  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300

Tron Y. Fu  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300

Daniel F. Gelwicks  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300

Christopher A. Suarez  
STEPTOE & JOHNSON LLP  
1300 Connecticut Avenue, NW  
Washington, DC 20036

Timothy Devlin  
Devlin Law Firm LLC  
1526 Gilpin Avenue  
Wilmington, DE 19806

Telephone: (202) 429-3000

Telephone: (302) 449-9010

5. The title and number of any case known to counsel to be pending in this or any other court or agency that will directly affect or be directly affected by this court's decision in the pending appeal:

*Express Mobile, Inc. v. Facebook, Inc.*, U.S. Court of Appeals for the Federal Circuit Case No. 2023-1645 , Patent Trial and Appeal Board IPR2021-01224; and *Express Mobile, Inc. v. Facebook, Inc.*, U.S. Court of Appeals for the Federal Circuit Case No. 2023-1646, Patent Trial and Appeal Board IPR2021-01226.

6. Organizational Victims and Bankruptcy Cases. Provide any information required under Fed. R. App. P. 26.1(b) (organizational victims in criminal cases) and 26.1(c) (bankruptcy case debtors and trustees). Fed. Cir. R. 47.4(a)(6).

Not Applicable.

Dated: November 13, 2023

/s/ James R. Nuttall

James R. Nuttall  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300  
Facsimile: (312) 577-1370

*Counsel for Plaintiff-Appellant,  
Express Mobile, Inc.*

## TABLE OF CONTENTS

EXEMPLARY PATENT CLAIMS ILLUSTRATIVE OF THE ISSUES.....	i
CERTIFICATE OF INTEREST FOR EXPRESS MOBILE, INC.....	ix
STATEMENT OF RELATED CASES .....	1
STATEMENT OF JURISDICTION.....	1
STATEMENT OF THE ISSUES.....	2
STATEMENT OF THE CASE AND FACTS .....	2
I. '397 PATENT FAMILY.....	3
A. Technology of the '397 Patent Family .....	3
B. Accused Products .....	4
C. Claim Construction.....	5
1. Judge Andrews's Claim Construction in <i>eGroves</i> .....	6
2. Judge Seeborg's Claim Construction in <i>X.COMMERCE</i> .....	6
3. Judge Andrews's Claim Construction in <i>Shopify</i> .....	7
4. Judge Andrews's Claim Construction in This Litigation.....	7
D. Summary Judgment .....	9
E. Motion for Reargument .....	9
II. '755 PATENT FAMILY .....	9
A. Technology of the '755 Patent Family .....	9
B. Claim Construction and Summary Judgment.....	10
C. Trial .....	11
1. Motion in <i>Limine</i> No. 6 .....	11
2. Trial.....	12
D. Judgment as a Matter of Law or New Trial .....	13
SUMMARY OF ARGUMENT .....	14
ARGUMENT .....	15
I. STANDARD OF REVIEW .....	15



II. '397 PATENT FAMILY .....	18
A. THE DISTRICT COURT ERRED IN ITS CONSTRUCTION OF RUNTIME ENGINE .....	18
1. "Runtime Engine" Should Be Construed With "Utilize Information" Language.....	18
2. The Limited Support For "Reading Information" Fails. ....	23
3. Under The Correct "Utilizing" Construction, Summary Judgment Should Be Reversed.....	26
B. THE DISTRICT COURT ERRED IN GRANTING SUMMARY JUDGMENT OF NON-INFRINGEMENT APPLYING ITS CONSTRUCTION OF RUNTIME ENGINE.....	27
1. The RTE3 JavaScript Files Read Information From GoDaddy's Database And Do Not Rely On Intermediate Files. ....	28
2. Use Of An API To Read From A Database Is A Genuine Dispute Of Material Fact Precluding Summary Judgment. ....	31
3. Direct Reading Is Not Required. ....	35
III. '755 PATENT FAMILY .....	40
A. THE DISTRICT COURT ERRED IN DENYING JUDGMENT AS A MATTER OF LAW AND A NEW TRIAL .....	40
1. The District Court Erred In Finding That GoDaddy Did Not Contradict The Court's Claim Construction At Trial.....	40
2. The District Court Erred In Finding There Were Alternative Bases For Non-Infringement.....	55
3. The District Court Erred In Denying A New Trial.....	58
CONCLUSION AND STATEMENT OF RELIEF SOUGHT .....	60

## TABLE OF AUTHORITIES

	<b>Page(s)</b>
<b>Cases</b>	
<i>Absolute Software, Inc. v. Stealth Signal, Inc.</i> , 659 F.3d 1121 (Fed. Cir. 2011) .....	16, 38, 39
<i>Accent Packaging, Inc. v. Leggett &amp; Platt, Inc.</i> , 707 F.3d 1318 (Fed. Cir. 2013) .....	26
<i>Ancora Techs., Inc. v. Apple, Inc.</i> , 744 F.3d 732 (Fed. Cir. 2014) .....	23
<i>Apple Inc. v. Samsung Elecs. Co.</i> , 839 F.3d 1034 (Fed. Cir. 2016) .....	16
<i>Avid Tech., Inc. v. Harmonic, Inc.</i> , 812 F.3d 1040 (Fed. Cir. 2016) .....	17, 59
<i>Cardiac Pacemakers, Inc. v. St. Jude Med., Inc.</i> , 381 F.3d 1371 (Fed. Cir. 2004) .....	17
<i>Cordis Corp. v. Boston Sci. Corp.</i> , 561 F.3d 1319 (Fed. Cir. 2009) .....	24
<i>Cordis Corp. v. Boston Sci. Corp.</i> , 658 F.3d 1347 (Fed. Cir. 2011) .....	50, 51
<i>Draper v. Airco, Inc.</i> , 580 F.2d 91 (3d Cir. 1978) .....	59
<i>Enercon GmbH v. International Trade Com’n</i> , 151 F.3d 1376 (Fed. Cir. 1998) .....	24
<i>Eon Corp. IP Holdings LLC v. Silver Spring Networks, Inc.</i> , 815 F.3d 1314 (Fed. Cir. 2016) .....	49, 52
<i>ePlus, Inc. v. Lawson Software, Inc.</i> , 700 F.3d 509 (Fed. Cir. 2012) .....	52
<i>Evolusion Concepts, Inc. v. HOC Events, Inc.</i> , 22 F.4th 1361 (Fed. Cir. 2022) .....	26

<i>Exergen Corp. v. Wal-Mart Stores, Inc.</i> , 575 F.3d 1312 (Fed. Cir. 2009) .....	40
<i>Finisar Corp. v. DirecTV Grp., Inc.</i> , 523 F.3d 1323 (Fed. Cir. 2008) .....	53, 54
<i>GE Lighting Solutions, LLC v. AgiLight, Inc.</i> , 750 F.3d 1304 (Fed. Cir. 2014) .....	24, 25
<i>Graphic Packaging Intl., Inc. v. C.W. Zumbiel Co.</i> , No. 3:10-cv-891-J-37 JBT, Slip Op. (M.D. Fla. July 16, 2012).....	50
<i>Hill-Rom Servs., Inc. v. Stryker Corp.</i> , 755 F.3d 1367 (Fed. Cir. 2014) .....	24
<i>Ingenio, Filiale De Loto-Quebec, Inc. v. Gamellogic, Inc.</i> , 445 F. Supp. 2d 443 (D. Del. 2006).....	39
<i>Inventio AG v. ThyssenKrupp Elevator Americas Corp.</i> , 5 F. Supp. 3d 665 (D. Del. 2013).....	39
<i>LaserDynamics Inc. v. Acer Am. Corp.</i> , No. CIV.A. H-01-1745, 2003 WL 25782746 (S.D. Tex. June 12, 2003) .....	39
<i>Lightning Lube, Inc. v. Witco Corp.</i> , 4 F.3d 1153 (3d Cir. 1993) .....	16
<i>Mannatech, Inc. v. Glycoproducts Int'l, Inc.</i> , No. 3-06-CV-0471-BD, 2008 WL 2704425 (N.D. Tex. July 9, 2008) .....	47, 50
<i>Mformation Techs, Inc. v. Research in Motion Ltd.</i> , 764 F.3d 1392 (Fed. Cir. 2014) .....	51
<i>Moba, B.V. v. Diamond Automation, Inc.</i> , 325 F.3d 1306 (Fed. Cir. 2003) .....	41, 49, 51
<i>Network-1 Techs., Inc. v. Hewlett-Packard Co.</i> , 981 F.3d 1015 (Fed. Cir. 2020) .....	17, 58, 59
<i>Nicini v. Morra</i> , 212 F.3d 798 (3d Cir. 2000) .....	16

<i>NobelBiz, Inc. v. Global Connect, L.L.C.</i> , 701 F. App'x 994 (Fed. Cir. 2017) .....	49
<i>Omega Patents, LLC v. CalAmp Corp.</i> , 920 F.3d 1337 (Fed. Cir. 2019) .....	58, 59
<i>Oracle Corp. v. Parallel Networks, LLC</i> , 375 F. App'x 36 (Fed. Cir. 2010) .....	37
<i>Personal Audio, LLC v. Google LLC</i> , No. CV-17-1751-CFC, 2023 WL 5723453 (D. Del. Sept. 5, 2023) .....	48, 50
<i>Pitney Bowes, Inc. v. Hewlett-Packard Co.</i> , 182 F.3d 1298, 51 USPQ2d 1161 (Fed.Cir.1999) .....	34
<i>Power Integrations, Inc. v. Fairchild Semiconductor Int'l, Inc.</i> , 843 F.3d 1315 (Fed. Cir. 2016) .....	17
<i>Rembrandt Wireless Techs., LP v. Samsung Elecs. Co.</i> , 853 F.3d 1370 (Fed. Cir. 2017) .....	52
<i>Shopify Inc. v. Express Mobile, Inc.</i> , CA No. 19-439-RGA, 2020 WL 3432531 (D. Del. June 23, 2020) .....	42, 43
<i>Shopify Inc. v. Express Mobile, Inc.</i> , No. CV 19-439-RGA, 2021 WL 4288113 (D. Del. Sept. 21, 2021) .....	30
<i>Teva Pharm. Indus. Ltd. v. AstraZeneca Pharm. LP</i> , 661 F.3d 1378 (Fed. Cir. 2011) .....	16
<i>Teva Pharmaceuticals USA, Inc. v. Sandoz, Inc.</i> , 574 U.S. 318 (2015) .....	15
<i>Vitronics Corp. v. Conceptronic, Inc.</i> , 90 F.3d 1576 (Fed. Cir. 1996) .....	23, 24, 26, 36
<i>Wi-LAN USA, Inc. v. Ericsson, Inc.</i> , 675 F. App'x 984 (Fed. Cir. 2017) .....	34
<i>Wi-LAN, Inc. v. Apple, Inc.</i> , 811 F.3d 455 (Fed. Cir. 2016) .....	51

**Other Authorities**

FED. R. CIV. P. 50 .....	2, 13
FED. R. CIV. P. 59 .....	13

## **STATEMENT OF RELATED CASES**

Pursuant to Federal Circuit Rules 28(a)(4) and 47.5, counsel for Plaintiff-Appellant Express Mobile, Inc. identifies the following related cases known to counsel that will directly affect or be directly affected by this Court's decision in the present appeal:

*Express Mobile, Inc. v. Facebook, Inc.*, U.S. Court of Appeals for the Federal Circuit Case No. 2023-1645, Patent Trial and Appeal Board IPR2021-01224; and *Express Mobile, Inc. v. Facebook, Inc.*, U.S. Court of Appeals for the Federal Circuit Case No. 2023-1646, Patent Trial and Appeal Board IPR2021-01226.

## **STATEMENT OF JURISDICTION**

The district court had jurisdiction over this case under 28 U.S.C. §§ 1331, 1338, and 2201. On August 8, 2022, the district court issued its memorandum opinion and order, granting Defendant-Appellee's motion for summary judgment of non-infringement as to two of five asserted patents, U.S. Patent Nos. 6,546,397 ("397 patent") and 7,594,168 ("168 patent"). The district court held a five-day jury trial, and on March 3, 2023, the jury returned a verdict of non-infringement as to the other three of five asserted patents, U.S. Patent No. 9,063,755 ("755 patent"), 9,471,287 ("287 patent"), and 9,928,044 ("044 patent"). On March 6, 2023, the district court entered final judgment after verdict of non-infringement as to the asserted '397, '168, '755, '287, and '044 patents. On April 3, 2023, Plaintiff-

Appellant timely moved under FED. R. CIV. P. 50 for renewed judgment as a matter of law or, in the alternative, for a new trial. On July 5, 2023, the district court denied Plaintiff-Appellant's motion for renewed judgment as a matter of law or, in the alternative, for a new trial. On August 3, 2023, Plaintiff-Appellant timely filed its notice of appeal. This Court has jurisdiction over this appeal under 28 U.S.C. § 1291.

### **STATEMENT OF THE ISSUES**

1. Whether the district court erred in determining on summary judgment that Defendant did not infringe the '397 and '168 patents, including the claim construction of "runtime engine."

2. Whether the district court erred in entering judgment after verdict of non-infringement as to the '755, '287, and '044 patents and denying Plaintiff's renewed motion for judgment as a matter of law and/or new trial.

### **STATEMENT OF THE CASE AND FACTS**

Plaintiff-Appellant Express Mobile, Inc. ("Express Mobile" or "XMO") is a leader in the business of developing mobile applications and website design and development platforms, and has intellectual property including U.S. patents relating to certain tools useful in the field. Appx509 at ¶ 44. Steve Rempell, the former CEO and CTO of XMO, is an inventor on XMO's patent portfolio. *Id.* Mr. Rempell has over 50 years of experience in technology companies, with much of that work

focused on web-based technologies and applications. *Id.* Over the past several years, XMO has undertaken an initiative to safeguard its intellectual property through a series of legal actions. Appx5181–5184. XMO brought this lawsuit against Defendant-Appellee GoDaddy.com LLC (“GoDaddy” or “GD”) for infringement of five patents: ’397, ’168, ’755, ’287, and ’044 patents. Appx500 at ¶ 1; Appx1617 at ¶ 1. The first two patents (collectively, the “’397 Patent Family”) are directed to an apparatus and a method for building websites. Appx1618–1622 at ¶¶ 9–25. The remaining three patents (collectively, the “’755 Patent Family”) are directed to a system and a method for incorporating and displaying web services. Appx1622–1626 at ¶¶ 26–43.

## **I. ’397 PATENT FAMILY**

### **A. Technology of the ’397 Patent Family**

The ’397 Patent Family addresses the limitations of conventional web construction tools, which required users to program directly in HTML or use a desktop editor that output HTML-formatted files. Appx208 at 1:11–49. The ’397 Patent Family introduces a browser-based build engine and interface, offering a WYSIWYG (what you see is what you get) web development experience. *Id.* at 2:34–37. This involves user selectable settings that are stored in a database and a run time file. Appx240 at 65:58–60. The system stores the user-selected settings for the website, including content, layout, and other settings, into a database.



Appx240 at 65:58–59; Appx343 at 64:63–67. The run time file “utilizes information stored in [the] database” to generate specific commands, ensuring that the website is displayed correctly. *Id.* at 65:67. The system uses a runtime engine to dynamically generate the website, assembling web pages using the objects and style information extracted from the database. Appx240 at 65:64–66:2; Appx344 at 65:3–7.

Asserted claims 1, 2, 3, 11, and 37 of the ’397 patent each require a “run time file,” and claims 1, 2, and 3 of the ’168 patent each require a “runtime engine.” Appx050. The Parties agree that at least one run time file *includes* runtime engine. Appx3079 at 60:1–11 (GD’s expert); Appx4391 at 47:13–19 (GD’s counsel).

## **B. Accused Products**

GD offers subscription products referred to as Website Builder and Managed WordPress, which are accused of infringement in this litigation. Appx031. Both Website Builder and Managed WordPress build and generate websites by storing user-selected settings in GD’s database and using certain files as the runtime engine to utilize information from the database and create webpages. Appx5410–5412 at ¶ 313, Appx5417 at ¶ 325, Appx5426–5427 at ¶ 341, Appx5678–5683 at ¶¶ 784–790; Appx5994–5995 at ¶ 200. Specifically, Run Time Engine Category 3 (“RTE3”) includes JavaScript files sent to a user’s browser. Appx051.<sup>1</sup> It is undisputed that

---

<sup>1</sup> Run Time Engine Categories 1, 2, and 4 are other types of Run Time Engines that are not at issue on this Appeal. Appx053; Appx15191.

these RTE3 JavaScript files are downloaded to a user's web browser at runtime. Appx052; Appx4796 at ¶ 39; Appx5410–5412 at ¶ 313, Appx5417 at ¶ 325. These RTE3 JavaScript files utilize information from the database by making specific function calls (*i.e.*, by “fetching” information from the database). Appx5418–5426 at ¶¶ 329–339, Appx5434–5435 ¶¶ 350–354 (XMO's expert); Appx5994–5995 at ¶ 200, Appx6006 at ¶ 220; Appx10578 at ¶ 20 (XMO's expert); Appx4795 at ¶ 35; Appx11118 at 115:8–15; Appx11541 at 170:2–5. Specifically, the RTE3 JavaScript files fetch user-selected settings from the database using the function call `fetch()`. *Id.* GD's database communicates with the RTE3 JavaScript files through an Application Programming Interface (API), which is a part of the database itself and functions as a translator. *Id.*

### **C. Claim Construction**

The terms “runtime engine” and “run time file” have been construed previously in several related cases involving the '397 Patent Family: (1) *Express Mobile, Inc. v. eGroves Sys. Corp.*, No. 1:17-cv-00703 (D. Del.) (Andrews, J.) (“*eGroves*”); (2) *X.COMMERCE v. Express Mobile, Inc.*, No. 3:18-cv-03287 (N.D. Cal.) (Seeborg, J.) (“*X.COMMERCE*”); and (3) *Shopify Inc. v. Express Mobile, Inc.*, No. 1:19-cv-00439 (D. Del) (Andrews, J.) (“*Shopify*”).

### 1. Judge Andrews’s Claim Construction in *eGroves*

In *eGroves*, both XMO and eGroves proposed a construction of “runtime engine” such that a file when executed at runtime “facilitates the retrieval of information from the database.” Appx1841. Judge Andrews subsequently agreed with this construction, holding that runtime engine is a file executed at runtime that “*facilitates the retrieval of information* from the database and generates commands to display a web page or website.” Appx1801.<sup>2</sup>

### 2. Judge Seeborg’s Claim Construction in *X.COMMERCE*

In *X.COMMERCE*, XMO maintained a consistent construction of “runtime engine” as in *eGroves*, proposing “runtime engine” such that a file when executed at runtime “*facilitates the retrieval of information* from the database.” Appx1830. However, X.COMMERCE proposed that a “runtime engine” is a file when executed at runtime “*reads information* from the database.” Appx1830. Judge Seeborg construed the term to mean “[a] file that is executed at runtime that *reads information from the database* and generates commands to display a web page or website.” Appx21781. Judge Seeborg stated that “facilitates retrieval of information from the database” “introduce[d] unwarranted ambiguity not grounded in anything expressly described in the specification.” Appx21780.

---

<sup>2</sup> All emphasis added unless otherwise noted.

### 3. Judge Andrews’s Claim Construction in *Shopify*

In *Shopify*, XMO consistently maintained the same construction for the term “runtime engine” as it did in *eGroves* and *X.COMMERCE*. Specifically, XMO proposed that a “runtime engine” should be construed as a file that, when executed at runtime, “*facilitates the retrieval of information* from the database.” Appx1850. In contrast, Shopify suggested that the term should be construed as “[a] file that is executed at runtime that *reads information* from the database.” *Id.* Judge Andrews acknowledged that a “reading” requirement “appear[ed] in descriptions of embodiments,” and also noted that the prosecution history confirms that a runtime engine “can utilize information from the database in a number of ways, including by acting as a template shell that is combined with a database to generate the appropriate commands.” *Id.* Ultimately, Judge Andrews adopted Shopify’s construction, citing Judge Seeborg’s earlier inclination away from XMO’s proposed phrase “facilitates retrieval of information from the database” as vague. *Id.*

### 4. Judge Andrews’s Claim Construction in This Litigation

This case was initially assigned to Judge Andrews. Appx031. On June 1, 2021, Judge Andrews issued an order construing the disputed claim terms, including a “runtime engine.” Appx031; Appx016. In light of the concerns expressed by both Judge Seeborg and Judge Andrews regarding the vagueness of “facilitates retrieval of information from the database,” and considering Judge Andrews’ own

observation that intrinsic evidence supported the notion that a runtime engine “can utilize information from the database,” XMO proposed an alternative construction. Appx1756. XMO proposed that a “runtime engine” be construed as a file that, when executed at runtime, “*utilizes information* from the database” *Id.*

During the *Markman* hearing, XMO expressed concerns that “reads information from the database” was too narrow. Appx4394 at 50:15–21. XMO argued that its construction is consistent with the claim language and intrinsic record and, as such, the term should encompass a broader range of functions than simply “reading.” Appx4394 at 50:15–21. Specifically, XMO raised concerns that “reading” might exclude other functions, such as “downloading” (Appx4388 at 44:7–12) or situations where “something else can read it or it could be combined” (Appx4388 at 44:15–20). To address these concerns, Judge Andrews questioned GD about whether “downloading information from the database” fell within the scope of “reading information from the database,” to which GD responded that “downloading” was equivalent to “reading” over a network. Appx4390 at 46:14–24. In his order, Judge Andrews memorialized that concession and, without further deliberation between “utilizing” and “reading,” accepted GD’s construction. Appx017.

#### **D. Summary Judgment**

On November 17, 2021, the Parties filed a summary judgment motion. Appx031. The case was subsequently reassigned to Judge Matthew Kennelly. *Id.* GD moved for summary judgment of non-infringement based on the “runtime engine” limitations. Appx050–051. Judge Kennelly granted summary judgment with respect to the RTE3 JavaScript files, based on an alleged lack of evidence establishing that the RTE3 JavaScript files read *directly* from the database. Appx053–055. Instead, Judge Kennelly found that: (1) these files solely interacted with a Content Delivery Network (CDN), not with the alleged GD database; and (2) depended on other files to facilitate access to the database. Appx053–054.

#### **E. Motion for Reargument**

Following the court’s summary judgment ruling, XMO sought reargument on the basis that the court had misunderstood the record in relation to GD’s RTE3 JavaScript files. Appx15191. XMO’s argument centered on the court’s misapprehension that the RTE3 JavaScript files do in fact interact with and “read information from” GD’s database. *Id.* However, the court denied XMO’s motion. Appx21788.

## **II. ’755 PATENT FAMILY**

#### **A. Technology of the ’755 Patent Family**

The ’755 Patent Family relates to a system that allows the display of web service content on a device platform. Appx345 at Abstract. An authoring tool is

employed to define a user interface (UI) objects for display, each UI object corresponding to a selected web component. Appx382 at 37:15–20, 38:26–31. The invention includes generating software code, including an Application, which is a device-independent code, and a Player, which is a *device-dependent code*. *Id.* at 37:26–30, 38:36–40. The system includes receiving inputs and the Player receiving and displaying output values. *Id.* at 37:31–46, 38:41–56. Notably, the ’755 Patent Family expressly claims embodiments of Player code that run within web browsers. Appx366 at 5:27–28. For example, dependent claim 13 of the ’287 patent in the ’755 Patent Family recites “[t]he system of claim 1, where said Player is activated and *runs in a web browser*” Appx424 at 39:10–11. GD’s Website Builder and Managed WordPress products are also accused of infringing the ’755 Patent Family.

## **B. Claim Construction and Summary Judgment**

The Parties agreed and the court entered a construction of “device-dependent code,” as “code that is specific to the operating system, programming language, or *platform* of a device.” Appx010. Judge Andrews’ construction of “device-dependent code” in this case mirrors the same definition applied in *Shopify*. Appx1859. Notably, in *Shopify*, Judge Andrews further recognized that the term “platform” was not strictly limited to operating systems, and acknowledged that “*browsers* could be the platform.” Appx6507; Appx6516. In both this case and *Shopify*, browser platforms were *not* excluded from the construction of “device-

dependent code.” Appx010; Appx1859. Additionally, the term “*platform*” was not construed in either case. *Id.* In both cases, both Judge Andrews and Judge Kennelly denied summary judgment because XMO identified files with conditionally branched JavaScript code related to a device’s browser platform as the Player. Appx056.

### **C. Trial**

#### **1. Motion in *Limine* No. 6**

Prior to trial, XMO filed a motion in *limine* to preclude evidence and arguments contrary to the court’s claim construction and summary judgment rulings. Appx15237–15240. GD’s expert asserted that “device-dependent code” must exclusively operate on a single type of device. Appx15238; Appx11398–11399 at ¶¶ 601–603. XMO contended that GD was attempting to rewrite the construction of a “Player.” Appx15238. XMO argued that GD was attempting to exclude Players that incorporated conditionally branched JavaScript code, a type of code that is specific to a device platform, from the court’s adopted construction. *Id.*

During the pretrial conference, XMO reiterated its concern that GD might advance the argument that “a separate player must be created for every single possible device.” Appx17002 at 50:3–7. In response to XMO’s apprehension, Judge Kennelly swiftly granted the motion, characterizing it “as a no-brainer.” Appx17001 at 49:7–14. GD then assured the court that “GoDaddy does not intend to argue that



a separate player must be provided for every single device . . .” Appx17005 at 53:2–6. Furthermore, Judge Kennelly stressed: “it just needs to be clear . . . that witnesses can’t opine about the proper construction of the claim. They have to adopt the construction of the claim that’s been ordered or found, and they have to go with that, so that’s kind of a truism.” Appx17007 at 55:12–17.

## 2. Trial

Despite the assurance made during the pretrial conference, GD and its expert contradicted the court’s adopted construction by unilaterally narrowing the construction. *See, e.g.*, 17455 at 629:9–10 (GD’s expert insisting that every instance of Player code must be coded to a specific operating system of a device), Appx17454 at 628:13 (“it’s the operating system”), Appx17454–17455 at 628:25–629:1 (“it was the operating system”). GD explicitly contended that “device specific means code designed for a particular device,” and that Player code “must be coded to account for the device’s *operating system*” and “obviously by device, it means operating system.” Appx17452 at 619:5–6 (GD’s counsel); Appx17467 at 679:12–14 (GD’s expert). GD disregarded the broader court-approved construction, which encompasses *operating systems, programming languages, and platforms*. Appx6506–6507; Appx010. Furthermore, GD’s arguments also incorrectly asserted that Player code could not run in web browsers, despite explicit claim language to the contrary. Appx17457 at 636:19, *id.* at 637:4–9. GD also suggested to the jury

that browsers could not fulfill the role of the “platform” in the Court’s construction of “device-dependent code.” Appx17472 at 698:18–699:1 (GD’s expert asserting that “I don’t see how [web browser]” is part of the grouping relating to the device of operating system, programming language, or platform related to that device).

**D. Judgment as a Matter of Law or New Trial**

XMO moved for judgment as a matter of law under FED. R. CIV. P. 50(a), and renewed its Rule 50 motion and moved for new trial under FED. R. CIV. P. 59. Appx120. At the heart of XMO’s argument was the contention that GD’s assertions contradicted the court’s claim construction of “device-dependent code.” Appx21682. XMO maintained that GD’s position improperly narrowed the definition to code exclusive to a device’s operating system, effectively excluding the “programming language, or platform” aspects of the court’s construction. Appx21689–21691. XMO further asserted that, based on the court’s construction, a web browser should be regarded as a “platform,” thereby classifying its code as device-dependent and, consequently, a Player. Appx21691–21692. However, the court, in response to XMO’s arguments noted that it had not provided a specific construction for the term “platform” and that the jury was “free to rely on the plain and ordinary meaning.” *Id.*

## SUMMARY OF ARGUMENT

The district court erred in its summary judgment decision as to the '397 Patent Family and in its post-trial JMOL and new trial decision as to the '755 Patent Family.

1. The district court's summary judgment of non-infringement for the '397 Patent Family hinged on its construction, which required "reading information from a database" for the claimed "runtime engine." However, the "reading" construction was erroneous and the proper construction should have been "utilizing information from a database." The "reading" construction improperly imports embodiments from the specification and ignores not only the actual language of the claims—every claim bearing on this issue in the '397 Patent Family recites "utilizing information"—but also the fact that the "utilizing" claim language was added and relied upon by the applicant in securing allowance of the claims. Moreover, even applying the court's construction of "reading information from the database," the district court failed to draw reasonable inferences in XMO's favor. Specifically, the accused RTE3 JavaScript files include the crucial "fetch()" function call, which reads information from GD's database. Reading information from the database through an API is the interface that both parties' experts admit is the typical, if not the only, way to read from a database. Moreover, there was no basis for the district court's requirement that GD's system directly read from the database when the claim construction does not impose such a narrow requirement. Finally, the court erred in

finding that the runtime engine obtained information only from a third-party CDN database and not from any GD database, when information is indeed obtained from GD's database.

2. The district court's denial of JMOL and a new trial as to the '755 Patent Family was based on a mistaken interpretation of GD's trial arguments and evidence. The court wrongly perceived GD's arguments and evidence about the claim scope as those of non-infringement. In particular, GD improperly and pervasively offered evidence and argument that browsers cannot be platforms within the district court's construction of the claimed "Player." By taking such positions regarding claim scope, GD contradicted the court's claim construction, effectively reading out part of the scope of the court's construction. The district court erred in condoning GD's contradictory claim construction at trial and failing to apply the full scope of its own claim construction in making the JMOL ruling.

## **ARGUMENT**

### **I. STANDARD OF REVIEW**

This Court reviews claim construction *de novo*. *Teva Pharmaceuticals USA, Inc. v. Sandoz, Inc.*, 574 U.S. 318, 331 (2015). The district court's ultimate construction of disputed terms, its findings based on intrinsic evidence, and the application of subsidiary factual findings to its claim construction analysis are reviewed *de novo*. *Id.*

This Court reviews the grant of summary judgment under regional circuit law. *Teva Pharm. Indus. Ltd. v. AstraZeneca Pharm. LP*, 661 F.3d 1378, 1381 (Fed. Cir. 2011). Under Third Circuit law, summary judgment rulings are reviewed *de novo*. *Nicini v. Morra*, 212 F.3d 798, 805 (3d Cir. 2000). The record is reviewed in the light most favorable to and all reasonable inferences are drawn in favor of the non-movant. *Id.* at 805–06. Ultimately, infringement is a question of fact. *Absolute Software, Inc. v. Stealth Signal, Inc.*, 659 F.3d 1121, 1129–30 (Fed. Cir. 2011). “On appeal from a grant of summary judgment of non-infringement,” this Court “determine[s] whether, after resolving reasonable factual inferences in favor of the patentee, the district court correctly concluded that no reasonable jury could find infringement.” 659 F.3d at 1130.

This Court reviews an order denying “JMOL under the standard applied by the regional circuit.” *Apple Inc. v. Samsung Elecs. Co.*, 839 F.3d 1034, 1040 (Fed. Cir. 2016). In the Third Circuit, JMOL is proper only if “viewing the evidence in the light most favorable to the nonmovant and giving it the advantage of every fair and reasonable inference, there is insufficient evidence from which a jury reasonably could find” for the nonmovant. *Lightning Lube, Inc. v. Witco Corp.*, 4 F.3d 1153, 1166 (3d Cir. 1993). “The question is not whether there is literally no evidence supporting the party against whom the motion is directed but whether there is evidence upon which the jury could properly find a verdict for that party.” *Id.* A

party seeking to disturb the non-infringement judgment “must establish that [the challenged jury] instructions were legally erroneous, and that the errors had prejudicial effect.” *Network-1 Techs., Inc. v. Hewlett-Packard Co.*, 981 F.3d 1015, 1022 (Fed. Cir. 2020) (citations omitted). “It is well established that when an incorrect jury instruction—such as an incorrect claim construction—removes from the jury a basis on which the jury could reasonably have reached a different verdict, the verdict should not stand.” *Cardiac Pacemakers, Inc. v. St. Jude Med., Inc.*, 381 F.3d 1371, 1383 (Fed. Cir. 2004); *see also Avid Tech., Inc. v. Harmonic, Inc.*, 812 F.3d 1040, 1042 (Fed. Cir. 2016).

Regional circuit law also governs the new trial standards of review. *Power Integrations, Inc. v. Fairchild Semiconductor Int’l, Inc.*, 843 F.3d 1315, 1326 (Fed. Cir. 2016). The Third Circuit reviews the denial of a new trial motion by applying the “abuse of discretion” standard. *Id.*

## II. '397 PATENT FAMILY

### A. THE DISTRICT COURT ERRED IN ITS CONSTRUCTION OF RUNTIME ENGINE

#### 1. “Runtime Engine” Should Be Construed With “Utilize Information” Language.

The district court construed the claimed “runtime engine” as a “run time file”<sup>3</sup> with two requirements: (1) it must be “downloaded or created when a browser is pointed to a web page or website” and (2) it must “read[] information from the database.” Appx027. The second requirement was the basis for summary judgment in this case. During claim construction as to that second requirement, XMO urged the district court to construe the term to mean “facilitates [or facilitating] the retrieval of information” instead of “reads [or reading] information.” Appx1756–1757. The only basis for rejecting “facilitates the retrieval of information” in the district court below and in prior proceedings was that the “facilitates” language was allegedly vague and not grounded in anything. Appx1831; Appx21780; Appx1850. Accordingly, XMO offered “utilizes [or utilizing] information,” which is firmly grounded in the explicit language of the claims, as an alternative to the “facilitates” language in an effort to address the Court’s concerns. Appx1756–1757.

---

<sup>3</sup> Claims 1, 2, 3, 11, and 37 of the '397 patent recite a “runtime file.” Claims 1, 2, and 3 of the '168 Patent and claim 24 of the '397 patent recite a “runtime engine.” There is no dispute that “at least one run time file” is construed to mean “one or more files, including a [runtime] engine, that are downloaded or created when a browser is pointed to a web page or website.” Appx027.

The district court acknowledged XMO’s argument that “‘utilizes information’ could mean something more than simply reading information from the database.” Appx016–017. However, the district court went on to dismiss the distinction between utilizing and reading simply because, as the district court acknowledged, GD had agreed at oral argument that “downloading is equivalent to reading over a network.” *See supra* SOC § I.C.4.<sup>4</sup> Ultimately, the district court did not explicitly address the “utilizes information” issue, but it intended not to overly narrow the term. Regardless, XMO asks this Court to reject the erroneous “reading” language and construe the claimed “runtime engine” to require “utilizing information from the database.”

The intrinsic evidence supports the propriety of the “utilizing information” construction. Indeed, the claims themselves provide overwhelming support. The claims expressly recite that “run time files *utilize* information stored” in databases. Appx240–242 at 65:66–67, 66:19–20, 69:1–2, 70:13–14 (claims 1, 2, 37, 39 of the ’397 patent); Appx4392 at 48:9–11 (GD agreeing “Yes, there is . . . the term utilizes information from the database in the ’397 claims”). “Reading,” by contrast, is nowhere to be found in the claims of the ’397 and ’168 patents. Specifically, each

---

<sup>4</sup> Throughout this brief, cross-references to sections of the Statement of the Case are identified with “SOC,” and cross-references to sections of the Argument are identified with “Argument.”



of the claims of the '397 patent that bears on the interaction of the runtime engine and the database uses the “utilizing” language:

“building one or more web pages to generate said website from at least a portion of said database and at least one run time file, where **said at least one run time file utilizes information stored in said database**” Appx240 at 65:64–67 ('397 patent, claim 1)

“a build tool having at least one run time file for generating one or more web pages, **said run time file operating to utilize information stored in said database**” Appx240 at 66:19–23 ('397 patent, claim 2)

“an external database containing data corresponding to said information stored in said internal database, and one or more run time files, where **said run time files utilize information stored in said external database**” Appx241–242 at 68:65–69:2 ('397 patent, claim 37)

“means for building one or more web pages for generating said website from at least a portion of said database and at least one run time file, where **said at least one run time file utilizes information stored in said database**” Appx242 at 70:10–14 ('397 patent, claim 39)

Claim 1 of the '168 patent is the only claim bearing on this issue that does not recite “utilizes information,” but its recitation is consistent with “utilizing” not “reading”:

“wherein the database is produced such that a web browser with access to a **runtime engine is configured to generate the web-site from the objects and style data extracted from the provided database**” Appx344 at 65:3–6 ('168 patent, claim 1)

Thus, the claims, in addition to explicitly using the “utilizes information” language, consistently focus on *where* the run time file obtains information from (*i.e.*, the database) not *how* the run time file obtains such information. There is no claim

support for limiting *how* the claimed run time file obtains information from the database (*i.e.*, via reading from the database).

While the specification—not the claims—does contain references to reading information from the database in particular embodiments (*e.g.*, Appx210 at 5:59–62), the term “utilize information” aligns well with these exemplary uses. For example, the specification discloses reading a database via downloading from a server as an embodiment and as shown in Figure 29. Appx230 at 45:56–57 (“If not, the [runtime] engine will *read* the ‘Websitename’.dta file directly from the server.”); *id.* at 45:44–56. The specification also discloses the typical operation of reading from a database through an API. *Id.*; *see also* Appx173 (“RUNTIME ENGINE **READS** A PARAM VALUE WHICH POINTS TO THE DATABASE AND **INITIATES THE READ OPERATION.**”); Express Mobile’s expert explained that such reading over a network (including by “downloading from a server with access to the database”) is reading information from a database and well-known. Appx5428 at ¶ 344; *see also* Appx5427 at ¶ 342 (XMO expert explaining that use of “database access codes” is “read[ing] information from a database”); Appx5434 at ¶ 351 (use of a Java Applet runtime engine is reading from a database). Accordingly, the specification’s references to “reading” do not limit how the information is obtained, and are instead consistent with utilizing information.

The prosecution history also overwhelmingly supports the “utilizes information” construction. The claims were amended in January 2002 after a final office action and the “utilize information” language was introduced at that time. Appx629–631. In addition, the applicant emphasized that a runtime engine can utilize information from the database in a number of ways, including by acting as a template shell that is combined with a database to generate the appropriate commands. Appx666. In distinguishing the invention over the prior art, the prosecution emphasized that, consistent with the claim language as amended, “the pending claims now recite that run time files *utilize information* stored in the database,” and reference “the *use* of the [runtime] engine to *use* database information.” Appx665. The applicant distinguished the present invention based on its separation of a runtime engine from a database of user settings—“[i]n contrast, the claimed invention generates web pages using two features: a [runtime] engine and a database of user settings.” *Id.* **How** the settings from the database are made available to the runtime engine was not necessary to distinguish the claimed “runtime engine” over the prior art and secure allowance during prosecution.

Prior to the introduction of the “utilizing” language to the claims, the applicant argued that the claims were not anticipated by the prior art at issue because the invention was “for the production of web pages.” Appx616. In making that argument, the applicant stated that the “websites are generated by the *runtime*

*engine reading* and interpreting the external database and then building the web pages dynamically.” *Id.* After and despite that statement, the applicant chose to amend the claims to recite the “utiliz[ing] information” language above, not “reading.” Appx629–631. And thereafter, the claims were allowed as amended. The examiner’s reasons for allowance confirm the propriety of the utilizing construction: “[the prior art references] do not show a run time file *utilizing* user selected setting stored in a database.” Appx4596.

## **2. The Limited Support For “Reading Information” Fails.**

Ultimately, any support for “reading information” is found in exemplary embodiments of the specification and limited prosecution history statements taken out of context. Such support falls far short of this Court’s exacting legal standards for importing embodiments from the specification or surrendering claim scope during prosecution.

In construing a claim term, the Federal Circuit looks to the words of the claim itself. *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582 (Fed. Cir. 1996). If the claim term has a plain and ordinary meaning, the inquiry ends. *Id.* Additionally, “a clear ordinary meaning is not properly overcome (and a relevant reader would not reasonably think it overcome) by a few passing references that do not amount to a redefinition[.]” *Ancora Techs., Inc. v. Apple, Inc.*, 744 F.3d 732, 738 (Fed. Cir. 2014). The Federal Circuit has “repeatedly stated that while claims

are to be construed in light of the specification, they are not necessarily limited by the specification.” *Enercon GmbH v. International Trade Com’n*, 151 F.3d 1376, 1384 (Fed. Cir. 1998); *see also GE Lighting Solutions, LLC v. AgiLight, Inc.*, 750 F.3d 1304, 1308–10 (Fed. Cir. 2014). Disavowal requires that the specification or prosecution history make clear that the invention does not include a particular feature, or is clearly limited to a particular form of the invention. *Hill-Rom Servs., Inc. v. Stryker Corp.*, 755 F.3d 1367, 1372 (Fed. Cir. 2014); *Cordis Corp. v. Boston Sci. Corp.*, 561 F.3d 1319, 1329 (Fed. Cir. 2009). A claim construction that excludes preferred embodiments, however, is “rarely, if ever, correct.” *See Vitronics Corp.*, 90 F.3d at 1583 (Fed. Cir. 1996).

Here, the “utiliz[ing] information” claim language is not unclear<sup>5</sup> and the term should be given its plain and ordinary meaning. Even if it was somehow unclear, the specification’s references to “reading” in specific embodiments do not provide a basis for narrowing the scope of the claims. As explained *supra*, the specification’s references to “reading” consistently establish that reading can occur through an API or through downloading through a server, both of which are consistent with the explicitly recited “utilizing information” where there is no limit on *how* the

---

<sup>5</sup> Appx2147–2148 at ¶ 60–61 (Express Mobile’s expert stating that “utilizes information” is “clear, consistent with the intrinsic record.”); Appx3072 at 31:2–18 (GoDaddy’s expert, “Q. Now, what does it mean to utilize information stored in a database? A. To use it.”)

information is obtained. *See supra* Argument § II.A.1. That interpretation is entirely consistent with the claim construction proceedings, where Judge Andrews acknowledged GD’s concession that “downloading” fell within the scope of “reading.” *See supra* SOC § I.C.4.

The prosecution history is also entirely consistent with “utilizing information” as recited in the claims and provides no basis for placing a limitation on *how* the runtime engine obtains information from the database. If anything, the amendment to recite “utilizing information” and subsequent applicant and examiner statements, which were made only after the very limited statement about “reading” (Appx616), evidences a clear intent to maintain the scope afforded by “utilizing” rather than limit the claim scope to “reading.” Appx629–631; Appx665; Appx4596. Thus, similar to the “reading” statements in the specification, the prosecution history does not rise to the level of limiting the claim scope to “reading information from a database.”

Ultimately, there is no basis for importing “reading information” from the specification embodiments or prosecution history. The district court erred in adopting the “reading” claim construction in the absence of proper support in the specification or prosecution history to stray from the “utilizing” claim language. *See GE Lighting Solutions, LLC*, 750 F.3d at 1308–10. Moreover, in ultimately finding summary judgment of non-infringement here as addressed *infra* § II.B, the district

court used its claim construction to exclude preferred embodiments. Specifically, the district court held that GD’s accused RTE3 JavaScript files, which obtain information from a database by downloading using an API, did not meet the “reading information from a database” construction. Appx053–055. That claim interpretation necessarily and improperly excludes preferred embodiments indicating that reading can occur by downloading or through an API. *See supra* Argument § II.A.1. (referencing Figure 29). The “reading information” construction should be rejected for that additional reason. *See Vitronics Corp.*, 90 F.3d at 1583.

### **3. Under The Correct “Utilizing” Construction, Summary Judgment Should Be Reversed.**

Reversal is proper where, as here, the summary judgment of non-infringement was based upon an erroneous claim construction. *See Evolusion Concepts, Inc. v. HOC Events, Inc.*, 22 F.4th 1361, 1367–68 (Fed. Cir. 2022) (reversing a summary judgment of non-infringement for two independent claims after correcting the district court’s claim construction); *Accent Packaging, Inc. v. Leggett & Platt, Inc.*, 707 F.3d 1318, 1326, (Fed. Cir. 2013) (reversing summary judgment of non-infringement due to district court’s claim construction error). Here, the grant of summary judgment was based upon the erroneous “reads information” construction. Appx053–054. In addition, ample evidence establishes that GD’s accused RTE3 JavaScript files do in fact “utilize information” from GD’s database under the correct construction. *E.g.*, Appx5426–5427 at ¶ 341, Appx5678–5683 at ¶¶ 784–790;

Appx5236–5239; Appx15215 (GD agreeing that the accused RTE3 JavaScript files do obtain data from the database); *see also supra* SOC § I.B. At a minimum, such evidence creates a genuine issue of material fact. Thus, reversal of summary judgment of non-infringement is appropriate.

**B. THE DISTRICT COURT ERRED IN GRANTING SUMMARY JUDGMENT OF NON-INFRINGEMENT APPLYING ITS CONSTRUCTION OF RUNTIME ENGINE**

Even if the construction “read information from a database” is applied, the district court erred in granting summary judgment of non-infringement of GD’s RTE3 JavaScript files. In particular, the district court found that GD’s RTE3 JavaScript files do not “read information from the database” because those files instead rely on other files to call upon the database and “interact only with the [third-party] CDN.” Appx053–054. Those findings were in error. There is no dispute that the accused RTE3 JavaScript files do obtain information from the database and, thus do not interact only with the CDN. That error alone supports the relief now sought, but the district court further erred in finding that the accused RTE3 JavaScript files rely on non-existent intermediate files to obtain database information. Moreover, there is a genuine dispute of material fact over whether use of the API that GD admits performs the reading meets the “read information from a database” limitation, even if the district court considered the API to be an intermediate file (which it is not).



Lastly, regardless of the foregoing errors, “direct reading” is not required and summary judgment was thus improper.

**1. The RTE3 JavaScript Files Read Information From GoDaddy’s Database And Do Not Rely On Intermediate Files.**

The district court erroneously found that the accused RTE3 JavaScript files interact only with the CDN and do not themselves read from GD’s database, instead relying on intermediate files. Appx153. The court’s finding was based on GD’s repeated attempts to confuse the court. GD argued that the RTE3 JavaScript files never directly read any information from a GD database because the files are not “themselves reading” but rather only the API can read GD’s database. Appx15207. There is no dispute, however, that the RTE3 JavaScript files themselves, using their aptly named “fetch()” function call, obtain information from the GD database. *See* SOC § I.B. No intermediate files are involved.

The accused RTE3 JavaScript files themselves read information from GD’s database through the “fetch()” function call that does the fetching from GD’s database. *Id.* Specifically, the “fetch()” function call of the RTE3 files, as its name implies, reads from the GD database through an API. *Id.* There is no dispute that the crucial “fetch()” function call is within the specifically accused RTE3 JavaScript files. *Id.* What’s more, nowhere did the district court or GoDaddy identify what those alleged intermediate files were. Instead, all the evidence consistently

establishes that the “fetch()” function call obtains information from the database. *Id.* Thus, the district court erroneously found that the RTE3 JavaScript files do not “actually read” information from the database because “those files instead rely on other files to call upon the database.” Appx053.

The district court likewise erred in finding that the accused RTE3 JavaScript files instead interact only with the CDN. *Id.* The record reflects that GD’s RTE3 JavaScript files read user selectable settings from GD’s database, not “only with the [third-party] CDN” as the Court found. Appx5410–5412 at ¶ 313, Appx5417 at ¶ 325, Appx5426–5427 at ¶ 341, Appx5678–5683 at ¶¶ 784–790; Appx5994–5995 at ¶ 200 (XMO’s expert citing script.js that utilizes information from the database). For example, GD’s witness, Mr. Silvas, did not dispute that RTE3 JavaScript files include function calls that fetch and ultimately download information from the database. Appx4795 at ¶ 35. In fact, GD’s entire argument and the district court’s findings discussed below regarding an API are ***only relevant to GD’s database*** and are irrelevant to the third party CDN. *See infra* § II.B.2. Thus, GD’s arguments and the court’s findings regarding reading a database through an API would be completely meaningless if the accused RTE3 JavaScript files only obtained information from a third party CDN. The court’s finding otherwise was an erroneous finding and alone supports reversal of the court’s summary judgment decision.

In finding that the accused RTE3 JavaScript files relied on intermediate files and interacted only with the CDN rather than reading from the database, the district court analogized GD's system to that at issue in *Shopify*. Appx053–054. But GD's system is not like that at issue in *Shopify*, and the court's reliance on the facts in *Shopify* to decide the GD case resulted in several erroneous findings and conclusions. In *Shopify*, the district court found on summary judgment that “the alleged runtime engine did not read the database directly but rather ‘triggering the drop file to perform its data fetching function.’” Appx053 (quoting *Shopify Inc. v. Express Mobile, Inc.*, No. CV 19-439-RGA, 2021 WL 4288113, at \*20 (D. Del. Sept. 21, 2021) (hereinafter “*Shopify Summary Judgment*” available at Appx6477)). Here, the accused RTE3 JavaScript files themselves contain the “fetch()” function call. *E.g.*, Appx6006 at ¶ 220. In GD's accused system, there is no second file that is triggered to perform the fetching function. There were no direct “fetch()” or other calls to the database in the accused RTE files in *Shopify* and there are no comparable intermediate files that are called to perform reading in this case. Thus, the district court erred in finding that XMO's position in the present case is “similar to” that of *Shopify* and that the “same logic applies here.” Appx053.

The district court also relied on GD's declarations from Mr. Jarret and Mr. Silvas to support its conclusion that “the accused [runtime] engines ‘do not read from or contain any references to the source of the data at all, whether through a database-

access library or otherwise.” Appx054 (citing Appx4811–4817, at ¶¶ 10–29 (Jarret) and Appx4789–4796, at ¶¶ 14–17, 21–30, 34–40 (Silvas)). The district court’s reliance on Mr. Jarret was entirely in error. His declaration related only to RTE 4 files, not the RTE3 JavaScript files at issue in that section of the district court’s opinion and subject to this appeal. Appx4811–4817 at ¶¶ 10–29. And while Mr. Silvas’ declaration initially suggested that GD’s RTE3 JavaScript files retrieve information only from a third party CDN, he later admitted that these files do get information from GD’s database. Appx4794–4795 at ¶¶ 31–35. Thus, those declarations fall far short of establishing that there is no genuine dispute of fact warranting summary judgment of non-infringement. The district court’s decision was therefore improper for at least these reasons and should be reversed.

## **2. Use Of An API To Read From A Database Is A Genuine Dispute Of Material Fact Precluding Summary Judgment.**

To the extent that the district court was referring to the use of an API as “intermediary files” or as otherwise not meeting the “reading from a database” limitation at issue, that is also in error because an API is part of the database—not an intermediary file—and using an API to read a database is the typical way to read a database and consistent with the specification. The API is not an intermediate structure—it is part of the database itself. Appx15195 (including diagram of same). As XMO’s expert explained: “Databases have APIs on them . . . it’s the mechanism by which data from a database is read.” Appx5863 at 322:9–11. APIs are simply

translators for the function call of GD's RTE3 files (here, "fetch()"), that is recited directly as code in the accused RTE3 files. Appx5413 at ¶ 314, Appx5443 at ¶ 376, Appx5460–5461 at ¶ 406, Appx5469 at ¶ 429, Appx5470 at ¶ 431; Appx5994–5995 at ¶ 200; Appx10576 at ¶ 14. Thus, as Dr. Almeroth opined, an RTE file retrieving information from a database through an API is "reading information from a database." Appx10576 at ¶ 14.

GD's experts admitted that the use of APIs is typical, if not the only way, to read from a database.

Witness	Evidence
Technical Expert, Mr. Kent	<p>Q. [] So is it your opinion that <b>in order to infringe a runtime engine, must call to an API</b> to look up a database?</p> <p>A. Well, it has to hit something. It has to request something from a database. So I suppose <b>typically would be done through an API.</b></p> <p style="text-align: center;">* * *</p> <p>Q. [] So if a <b>runtime engine calls a API to retrieve information from a database</b>, that would satisfy the requirement of <b>reading information from that database</b>, right?</p> <p>A. <b>Yeah, I guess so. . . .</b> I haven't thought about this. It has to call -- it has to call something from the database. That's the theory, and Dr. Almeroth is saying these files are doing that. What I'm saying is these [JavaScript render] files are not doing that regardless of whatever mechanism they might use the -- database.</p>

	Appx11540–11541 at 169:8–13, 169:25–170:5.
Technical Expert, Dr. Greenspun	<p>Q. Now, is it your opinion that <b>using an API to access a database</b> would allow one to <b>read from the database</b>?</p> <p>A. Yes . . . that’s the only way that – <b>the only practical way</b> that a Java program has to either <b>read from</b> or write to <b>a database</b>.</p> <p>Appx11118 at 115:8–15.</p>

GD weakly attempted to sidestep these admissions as lacking context. Appx15211–15212. But both GD experts admitted reading a database using an API was “the only practical way,” “typical[,],” and met the court’s claim construction of reading. GD attempts to run from these admissions by arguing that Mr. Kent said the GD RTE are not “doing that [using an API].” *Id.* But GD admits—in fact the entire premise of its summary judgment motion—is that the accused RTE uses an API. Appx053; Appx4795 at ¶ 35 (Mr. Silvas declaring that API is used). GD’s argument with respect to Dr. Greenspun is even weaker, arguing that he was talking about a different programming language (Java instead of JavaScript). Appx15211. But regardless of the programming language, Dr. Greenspun admitted a function call to a database using an API is reading from a database. Appx11540–11541 at 169:8–13, 169:25–170:11; Appx11118 at 115:8–15. Summary judgment of non-infringement because an API is used, despite GD’s admissions that using an API is

how reading from a database would “typically” be done, if not “the only practical way” it would be done, is clearly improper and should be reversed. Appx11540–11541 at 169:8–13, 169:25–170:11; Appx11118 at 115:8–15. And at the very least, given GD’s admissions that the “fetch()” function interacts with the API the dispute over whether the database encompasses the API and whether reading from a database using an API presents a factual issue that precludes summary judgment.

Summary judgment of non-infringement may only be granted if, after viewing the alleged facts in the light most favorable to the nonmovant and drawing all justifiable inferences in the nonmovant's favor, there is no genuine issue whether the accused device is encompassed by the patent claims. *Pitney Bowes, Inc. v. Hewlett–Packard Co.*, 182 F.3d 1298, 1304, 51 USPQ2d 1161, 1165 (Fed. Cir.1999). Even applying the reading construction, when reasonable inferences are drawn in XMO’s favor, including whether use of an API is reading and whether GD’s API is a part of the database, there is a dispute over whether the RTE3 JavaScript files read from the database. *See Wi-LAN USA, Inc. v. Ericsson, Inc.*, 675 F. App’x 984, 994 (Fed. Cir. 2017) (nonprecedential) (while affirming a narrow claim construction, vacating summary judgment of non-infringement because the district court disregarded conflicting evidence in the record and gave undue weight to the accused infringer’s evidence). Here, in granting summary judgment the district court erroneously overlooked XMO’s evidence indicating that the API is part of the database and

reading is performed using an API, instead giving undue weight to GD's expert opinion.

### **3. Direct Reading Is Not Required.**

Under the "reading information" construction, summary judgment of non-infringement was also in error because direct reading is not required. The district court agreed with GD's argument that the "fetch()" function does not meet the "read information from the database" limitation because it "rel[ies] on intermediary files" and does not read the database "*directly*" and granted summary judgment of non-infringement on that basis. Appx053. As set forth above, no intermediary files are invoked, and to the extent that reading through an API renders the reading "indirect," that is not a proper basis for summary judgment of non-infringement.

The claim construction simply requires "reading." It does not require "direct" reading. The court improperly further narrowed this claim term and construction by requiring "direct reading," thereby improperly excluding reading a database using an API, or the use of intermediary files (even if such intermediary files were at issue here). The intrinsic evidence, the claim construction proceedings, and expert testimony all support the conclusion that direct reading is not required. The specification includes preferred embodiments indicating that reading can occur via downloading or through an API. *See supra* Argument § II.A.1. (referencing Figure 29). There is certainly not "highly persuasive evidentiary support" for interpreting



the claims to exclude those preferred embodiments. *Vitronics* at 90 F.3d at 1583. And the broader intrinsic evidence, including the claims themselves and the prosecution history, repeatedly supports that interaction between the RTE and the database, whether the word “read” or “utilize” is considered, is such that the RTE obtains certain information from the database without limitation as to how the RTE obtains that information from the database. *See supra* Argument § II.A.1.

During claim construction, GD agreed that “downloading is equivalent to reading over a network” and the district court acknowledged that agreement in its claim construction opinion. *See supra* SOC § I.C.4. That is precisely what the “fetch()” function call is doing when it fetches through an API. *See, e.g.*, Appx6006 at ¶ 220 (XMO expert opining that “using the fetch function call that returns the information from the database” is “reading information from a database”); *see also supra* Argument § II.B.1. At a minimum, the district court acknowledged that “reading over a network” such as downloading, was within the scope of the construction and, in so doing, acknowledged that direct reading is not required. *See supra* SOC § I.C.4.<sup>6</sup> Appx1757. Moreover, there are multiple admissions that

---

<sup>6</sup> The lack of a requirement that reading from the database be direct is consistent with the term “database” itself. The parties agreed that the term “database” is “an electronic information storage system offering data storage and **retrieval**.” Appx1745–1748. That “retrieval” aspect is consistent with XMO’s positions and further supports the existence of a genuine dispute of material fact.

“reading from a database” can occur through use of an API. *See, e.g.*, Appx11118 at 115:8–15; *see also supra* Argument § II.B.1. And, more importantly, GD’s own technical experts admit that an API is the “only practical way” to read information from a database the file types at issue. *See supra* Argument § II.B.2.

The district court’s rejection of reading through an API runs counter to all of that overwhelming evidence. Any support that can be mustered to support the “reading” construction must give effect to this overwhelming evidence, all of which consistently do not require direct reading or exclude reading through an API. Requiring that the claimed reading be “direct” improperly limits the scope of the claims and must be rejected in view of that evidence.

In *Oracle Corp. v. Parallel Networks, LLC*, this Court vacated a district court’s summary judgment of non-infringement because the non-infringement basis addressed only one aspect of the construed scope and not the other. 375 F. App’x 36, 39 (Fed. Cir. 2010) (district court construed “releasing said Web server to process other requests” to mean “freeing the Web server to process other requests” and the district court found non-infringement based on a lack of freeing certain software, without addressing freeing hardware, which was within the scope of the construction). This Court went on to reject the accused infringer’s “narrow reading” of the claims because it was “not require[d]” by the claim construction and was not supported by intrinsic evidence. *Id.* Likewise, here, even if the “reading” claim

construction is applied, the construction does not require direct reading as evidenced in GD expert admissions and acknowledgment that reading over a network and downloading are within scope, and there is no support in the intrinsic evidence for limiting the term to direct reading. Like *Oracle Corp.*, a reasonable jury could find that the accused RTE3 JavaScript files meet the “reading” limitation by reading through an API.

In *Absolute Software*, this Court also vacated summary judgment of non-infringement because the district court failed to draw reasonable inferences in favor of the patentee, specifically that the accused functionality “itself” provided the claimed network communication link in view of undisputed facts. 659 F.3d at 1133. Similarly, here, the question at summary judgment was whether GD’s RTE3 JavaScript files “reads information from the database.” Both XMO and GD disputed whether the RTE3 JavaScript files actually read the database, and both sides submitted expert declarations on this point. It is undisputed that (1) RTE3 JavaScript files obtain user settings from GD’s database using the “fetch()” function call within those files (*e.g.*, Appx6006 at ¶ 220); (2) GD’s database communicates with the RTE3 JavaScript files through an API (*e.g.*, Appx15215); and (3) APIs are used to read databases (*e.g.*, Appx11118 at 115:8–15). Based on these undisputed facts, the logical inference is that the RTE3 JavaScript files themselves read information from

the database. Because a reasonable jury could find infringement on that basis, summary judgment was improper. *See Absolute Software, Inc.*, 659 F.3d at 1133.

Caselaw addressing the plain meaning of the term “read” confirms that direct reading is not required and supports the breadth of the term. *See Ingenio, Filiale De Loto-Quebec, Inc. v. Gamelogic, Inc.*, 445 F. Supp. 2d 443, 454–55 (D. Del. 2006) (The “ordinary meaning of the term ‘read,’ in the context of a computer processor” is “to input data from a storage device.”); *Inventio AG v. ThyssenKrupp Elevator Americas Corp.*, 5 F. Supp. 3d 665, 674 (D. Del. 2013) (agreeing that “reading” a signal means that the device in question “receives (i.e., gets)” the signal); *LaserDynamics Inc. v. Acer Am. Corp.*, No. CIV.A. H-01-1745, 2003 WL 25782746, at \*4 (S.D. Tex. June 12, 2003) (ordinary meaning of “read” is “to extract data from memory or a storage medium . . . .”). Here, the use of “read” in the specification,” the context of the claims reciting “utilizing information,” and the overall intrinsic focus on where the information is obtained from not how the information is obtained (*see supra* Argument § II.A.1.) is entirely consistent with the plain and ordinary meaning set forth in those cases. The district court’s grant of summary judgment based on a requirement of direct reading was erroneous for that additional reason.

### III. '755 PATENT FAMILY

#### A. THE DISTRICT COURT ERRED IN DENYING JUDGMENT AS A MATTER OF LAW AND A NEW TRIAL

The claimed invention relates to generating displays with interactive web services content and includes generating software code, including an Application, which is a device-independent code, and a Player, which is a *device-dependent code*. Appx382 at 37:26–30, 38:36–40. GD’s Website Builder and Managed WordPress include a JavaScript Player that includes code specific to the device platform, including the device *browser platform*. Appx17594 at 54:18–55:11; Appx17458 at 642:3–5; *see also* Appx17468 at 683:25–684:18. This code contains conditionally branched logic (*i.e.*, JavaScript code) for the device browser platform that executes certain code depending on the device platform and generates a display based on the detected browser type. Appx17368–17369 at 284:16–286:3. Simply put, one set of code is executed for one type of browser or device, while a different set of code is executed for another type of browser or device. Appx17368 at 285:17–23.

At trial, GoDaddy contradicted the district court’s constructions of “Player” and “device-dependent code” and JMOL of infringement and a new trial on that basis was erroneously denied.

#### 1. The District Court Erred In Finding That GoDaddy Did Not Contradict The Court’s Claim Construction At Trial.

As the district court below acknowledged, “[i]t is settled law that ‘[n]o party may contradict the court’s construction to a jury.’” Appx128 (quoting *Exergen*

*Corp. v. Wal-Mart Stores, Inc.*, 575 F.3d 1312, 1321 (Fed. Cir. 2009). Where a jury improperly imports its own limitations into the claim, a verdict based on the erroneous claim construction generally cannot stand. *Moba, B.V. v. Diamond Automation, Inc.*, 325 F.3d 1306, 1313–14 (Fed. Cir. 2003) (reversing district court’s denial of JMOL of infringement because “by allowing the jury to import an additional limitation into the claims, the district court fundamentally altered the verdict.”).

Here, the district court erred in finding that GD did not contradict its claim construction at trial. GD consistently argued and presented evidence that its browser detection code was not Player code because browsers cannot be platforms and because device-dependent code is limited to code specific to the operating system, and does not include code specific to a platform. *See supra* SOC § II.C.2. In so doing, GD effectively reargued claim construction during trial and read-out the term “platform” from the claim construction. Any purported application of the actual construction was merely generic, conclusory, and does not alter that repeated and improper narrow application. And GD’s contradiction, contrary to the district court’s finding, was very significant and warrants at least a new trial.

**a. There Can Be No Dispute That the District Court’s Player and Device-Dependent Code Constructions Include Browsers As A Platform.**

The district court’s claim construction makes clear that the claimed player as construed includes browsers as a platform. The term “Player” appears in all asserted claims of the ’755 Patent Family. “Player” was construed to mean “device-specific code which contains instructions of a device and which is separate from the Application.” Appx037; *see also Shopify Inc. v. Express Mobile, Inc.*, CA No. 19-439-RGA, 2020 WL 3432531, at \*5–\*6 (D. Del. June 23, 2020) (hereinafter “*Shopify Claim Construction*” available at Appx1846). “Device-specific code,” which was found to be interchangeable with “device-dependent code” was construed to mean “code that is specific to the operating system, programming language, or platform of a device.” *Shopify Claim Construction* at \*7 (available at Appx1852). The *Shopify* court selected that construction from the specification defining “device-specific” routines as “codes that are specific to the operating system, programming language, or platform of specific devices.” *Id.* (citing Appx365 at 3:58–62).

On summary judgment, the *Shopify* court specifically addressed the question of whether the term “player” includes browsers. There, Shopify argued that conditionally-branched code for different browser platforms did not meet the court’s claim constructions of Player and device-dependent code. Judge Andrews rejected those arguments. *Shopify Summary Judgment*,” at \*33 available at Appx6477

(“Shopify’s argument... fails.... It does not mean that conditional logic [for browser platforms of a device] cannot be a form of device-dependent code.”). Moreover, the *Shopify* court was well apprised of the precise scope issue for this Court’s consideration on appeal. Judge Andrews acknowledged that XMO’s argument was that “a browser is understood in the art as a ‘platform’ (and therefore meets my construction for ‘device-dependent code’).” *Id.* at \*32 (available at Appx6508). At the summary judgment hearing, XMO made its understanding of the scope of the court’s construction abundantly clear, stating that “[t]here is no dispute that a browser is a device platform.” Appx21873 at 86:7–8. That statement was met with no disagreement from the court or Shopify. In its summary judgment ruling, the court did not take issue with the arguments repeatedly made by both parties that the scope of the term included browsers. *Id.* (acknowledging arguments made by XMO (regarding “platforms of a device (such as a browser)” and Shopify (regarding the “scope of the term ‘Player’” encompassing “browser version and type” as a “device platform.”). Appx4016–4017. At no point did the *Shopify* court waiver in its interpretation of the scope of claim term as including browsers as the platform in the claimed Player.

GD, likewise, did not take issue with the *Shopify* court’s interpretation of the scope of the claimed Player. Instead, GD proposed and raised exactly the constructions of “Player” and “device-dependent code” that the *Shopify* court



provided. Appx1782 (specifically referring to the “same reasons” from the *Shopify* court for the construction of “Player” and characterizing the *Shopify* court as having “thoroughly analyzed” the issue). On summary judgment in the case below, XMO specifically stated that “[a] browser is a type of device platform” and cited the *Shopify* court’s acknowledgment of that scope. Appx9184 at n. 14. On reply, GD did not respond at all, let alone refute the status of browsers as a platform. Appx12202–12204. The district court in this case again denied summary judgment, finding that files that contain conditionally-branched code for browser platforms could be a Player and device-dependent code and, at the very least, created genuine issues of material fact. Appx056.

Thus, the scope of the claimed Player as previously construed includes browsers as the platform. That scope was acknowledged by Judge Andrews, Judge Kennelly, XMO, *Shopify*, and GD. Inconsistent with that repeatedly confirmed scope, GD proceeded to present evidence and argument that was fundamentally rooted in a narrower claim construction inconsistent with the already-decided and repeatedly confirmed scope of the claimed “Player.”

**b. GoDaddy Consistently Presented A Narrowed Claim Construction At Trial.**

Despite acknowledging the impropriety of presenting contradictory claim constructions to the jury, the district court erred in finding that GD did not violate those well-settled legal principles. The district court concluded that it was “not

persuaded that GD presented a modified or narrowed version of the Court’s claim construction to the jury.” Appx128–129. The district court further held that in view of the plain meaning of the terms “platform” and “database” the jury was free to conclude that “a browser is not a platform under the Court’s construction of device-dependent code (and therefore Player), and that a file is not a database under the Court’s construction of registry.” Appx129–130. Both findings were in error.

The district court fundamentally erred in recasting GD’s trial presentation about the *scope* of the claim terms as *factual evidence of non-infringement* applying the court’s claim construction. GD’s non-infringement position was not that its accused JavaScript browser detection code is not device-dependent code under the court’s claim construction; instead it contradicted the court’s claim constructions at trial by arguing that: (i) the court’s claim construction of “device-dependent code” is limited to only code specific to the operating system of a device (and therefore not to programming languages or platforms of a device), and (ii) the accused Player cannot relate to browsers at all, even though the claims of the patents expressly claim embodiments of Player code that run within web browsers. In so doing, GD presented a narrowed claim construction, not a non-infringement defense.

GD’s expert erased any doubt that the issue was one of claim scope (as opposed to a factual issue of infringement) when he agreed that he and Dr. Almeroth were “applying different definitions of device-dependent.” Appx17468 at 684:19–

23 (“I don’t agree with his definition.”). GD’s expert also argued that “device specific means code designed for a particular device,” and “obviously by device, *it means operating system.*” Appx17467 at 679:12–14, Appx17452 at 619:5–6. And when asked if he “typically” refers to browsers as platforms, Mr. Kent clarified that he had “never used that phrase in relation to browsers.” Appx17472 at 698:18–699:6. This testimony was directly contradictory to the court’s claim construction order and the order in *Shopify* adopted by the district court below, where the court explained that an “argument that device-dependent code must be developed and provided to specific devices ... fails” and that such a requirement “is not a claim limitation.” Appx056 (adopting the reasoning and citing to *Shopify Summary Judgment* at \*16–17 available at Appx6508–6510). GD nonetheless insisted that every instance of Player code must be coded to a specific operating system of a device. *See supra* SOC § II.D.2.

Injecting a requirement that the device-dependent code to be particular to a given operating system obviates the “or platform” language of the court’s construction—which is exactly what GD did at trial. For example, in addition to its expert’s improper testimony limiting device-dependent code only to operating systems, GD’s closing argument repeatedly referred to the “operating system” and “platform” interchangeably as though they were the same thing, and further improperly argued that “device-dependent code” solely related to the “the device’s

operating system [that] deal[t] with the device’s programming language and platforms.” Appx17611 at 1025:13–16; *see also*, e.g., Appx17611 at 1024:5, 1024:10, 1024:17, 1025:25 (using “operating systems” and “platforms” interchangeably). In closing, GD’s counsel argued that no browser falls within the scope of “platform” in the court’s claim construction and specifically acknowledged that a claim construction dispute existed:

Now, we have a little disagreement about platform because the Court's construction is clearly talking about it in the context of the device. It's the device, the programming language, operating system, or platform of the device. And we know they didn't invent or claim browsers, so we're talking about the platform of the device.

Appx17613 at 1035:12–20; *see also* Appx17615 at 1040:19–21.

Thus, GD presented a narrowed claim construction argument at trial. Specifically, GD limited the court’s construction of “device-dependent code” to operating systems and to specific devices, and excluded platforms and other language from the court’s construction to exclude browsers as a device platform. Such contradiction warrants JMOL of infringement. *See Mannatech, Inc. v. Glycoproducts Int’l, Inc.*, No. 3-06-CV-0471-BD, 2008 WL 2704425, at \*2 (N.D. Tex. July 9, 2008) (granting JMOL of infringement where the verdict was based on a “finding [that] is contrary to the court’s claim construction”).

XMO previewed its concerns that GD would contradict the claim construction during the pretrial conference. *See supra* SOC § II.D.1. During the conference, Judge Kennelly took GD’s representation that it would not advance the argument that a separate Player must be created for each and every possible device at face value. Appx17005 at 53:2–6. At trial, however, GD and its expert primarily presented testimony that aligned precisely with XMO’s initial prediction, contrary to their earlier assurance. Such misleading practices are a basis for vacating jury verdicts. *See Personal Audio, LLC v. Google LLC*, No. CV-17-1751-CFC, 2023 WL 5723453, at \*2–3, 13 (D. Del. Sept. 5, 2023) (vacating jury verdict and granting a new trial because a party, through its expert and counsel, advanced an argument contradictory to court’s claim construction despite representing that it would not do so).

Furthermore, the district court below itself characterized GD’s alleged factual non-infringement defense as an issue of claim scope. The district court stated that “Express Mobile is essentially raising a claim construction argument regarding the meaning of the terms ‘platform’ and ‘database.’” Appx129. Thus, the district court tacitly recognized that GD’s alleged factual defense rested upon a question of claim scope. It was an error to permit the jury’s verdict to stand where that verdict necessarily depended on an issue of claim scope.

This Court has reversed denial of JMOL of infringement where the jury was permitted to “import an additional limitation into the claims,” which “fundamentally altered the verdict.” *Moba, B.V.*, 325 F.3d at 1313–14. In *Moba, B.V.*, the jury had returned a non-infringement verdict to a method claim on the basis that the method applied by the accused machine did not sequentially perform the recited guiding steps, but rather simultaneously performed some of the steps, even though the district court had not construed the claim as requiring a sequential performance of the steps. *Id.* Here, GD similarly argued non-infringement on the basis that the accused conditionally-branched code for browser platforms cannot meet the claimed “platform” as construed, but, crucially, the court’s claim construction is not so narrow and does not exclude browsers as platforms.

Other decisions are in accord. In *Eon Corp. IP Holdings LLC v. Silver Spring Networks, Inc.* this Court found error where, instead of resolving the claim construction dispute, the district court essentially permitted the parties’ technical experts to argue the claim construction to the jury. 815 F.3d 1314 (Fed. Cir. 2016); *see also NobelBiz, Inc. v. Global Connect, L.L.C.*, 701 F. App’x 994 (Fed. Cir. 2017) (finding error where experts were permitted to argue to the jury what was the plain and ordinary meaning of the terms). That is precisely what happened here. As Mr. Kent explained, he and Dr. Almeroth disagreed on the “definitions.” Appx17468 at 684:19–23 (“I don’t agree with his definition.”). This Court and district courts have

consistently rejected expert testimony based on incorrect claim interpretations. *See e.g., Cordis Corp. v. Boston Sci. Corp.*, 658 F.3d 1347, 1358 (Fed. Cir. 2011); *Graphic Packaging Intl., Inc. v. C.W. Zumbiel Co.*, No. 3:10-cv-891-J-37 JBT, Slip Op. at 5 (M.D. Fla. July 16, 2012) (granting JMOL of infringement where expert improperly relied on “an unduly narrow and incorrect claim interpretation); *Mannatech*, 2008 WL 2704425 at \*5 (granting JMOL of infringement where non-infringement positions at trial were founded in a narrow interpretation contrary to the court’s claim construction); *Pers. Audio, LLC*, 2023 WL 5723453, at \*13 (vacating jury’s verdict because a party “effectively ignored the Court’s claim construction”).

GD’s alleged factual non-infringement positions at trial were, in reality, claim construction arguments that the claims are narrower than they were construed to be. The district court had already confirmed that “browsers could be the platform.” *See Shopify Summary Judgment* at \*19 (available at Appx6516). GD’s contradictory and narrow expert testimony and arguments at trial that browsers cannot be platforms fail accordingly.

**c. The District Court Erred In Refusing Express Mobile’s Request To Apply The Full And Inherent Meaning Of The Constructions.**

The district court further erred in faulting XMO for raising a claim construction argument on JMOL. Appx129. As explained in Argument § III.A.1.

*supra*, the full scope of “device-dependent code” as already construed includes browsers as platforms. Thus, XMO simply—and properly—seeks to enforce the full meaning inherent in the court’s claim construction, as it consistently did throughout the proceedings below.

In *Moba, B.V.*, this Court addressed arguments of waiver in a similar context. There this Court rejected waiver arguments where the patent holder was “seek[ing] enforcement of the trial court’s claim construction” not “to alter the district court’s claim construction on appeal.” 325 F.3d at 1314. XMO’s actions before the district court at trial and on JMOL were likewise proper. Likewise, a district court may not “alter[ ] the scope of the original construction” after trial, *Wi-LAN, Inc. v. Apple, Inc.*, 811 F.3d 455, 466 (Fed. Cir. 2016), it may “adjust constructions post-trial if the court merely elaborates on a meaning inherent in the previous construction.” *Mformation Techs, Inc. v. Research in Motion Ltd.*, 764 F.3d 1392, 1397-98 (Fed. Cir. 2014) (citing *Cordis*, 658 F.3d at 1356). It is permissible at the JMOL stage to “clarify[ ] a meaning *inherent* in the construction or mak[e] plain what should have been obvious to the jury.” *Wi-LAN*, 811 F.3d at 466. XMO’s request in this case—before, during, and after trial—was to apply the “meaning inherent in the previous construction.” See *Mformation*, 764 F.3d at 1397. In addition to applying that meaning throughout, the district court should have provided that necessary clarification at the JMOL stage. See *id.* at 1397–98.



The district court cites three cases (*ePlus*, *Rembrandt*, and *Eon*) for the proposition that in the absence of construction of a term, the jury was free to rely on the plain and ordinary meaning of the term. Appx129 (referencing terms “platform” and “database” as not construed). Those cases fail to provide a basis for the district court’s refusal to enforce its prior construction and find GD contradicted that construction. In *ePlus, Inc. v. Lawson Software, Inc.*, 700 F.3d 509 (Fed. Cir. 2012), the term with disputed plain and ordinary meaning was the claim limitation “determining.” *ePlus* did not involve the disputed plain and ordinary meaning of a term is part of the court’s construction of the claim limitation. Similarly, in *Rembrandt Wireless Techs., LP v. Samsung Elecs. Co.*, 853 F.3d 1370, 1376 (Fed. Cir. 2017), the term with disputed plain and ordinary meaning was the claim limitation “different types.” And, in *Eon Corp.*, 815 F.3d at 1317, the term with disputed plain and ordinary meaning was the explicit claim term.

By contrast here, “platform,” is *part* of the court’s claim construction of “device-dependent code” and the scope of that term was repeatedly confirmed in the proceedings below and in *Shopify*. The district court below permitted GD to present a narrow claim construction position contrary to its already confirmed scope under the guise of a non-infringement defense. That is exactly what *Eon* prohibits. See 815 F.3d at 1319–1320 (holding that the district court erred in not resolving this dispute but instead instructing the jury to apply the plain and ordinary meaning of

the terms, and permitting the parties’ technical experts to essentially argue the claim construction dispute to the jury).

**d. GoDaddy’s Narrowed Claim Construction Presentation Was Significant.**

The district court further erred in finding that any “difference between the allegedly incorrect constructions used by GoDaddy/Kent and the Court’s constructions” was not “significant enough” to entitle XMO to JMOL. Appx130. But here, the difference is crucial and effectively reads the word “platform” out of the construction. That difference was GD’s basis for non-infringement.

GD itself characterized the distinction as important at trial. For example, GD’s counsel argued in closing that the “key” is “Express Mobile knows this industry shift means it has to fit a square peg, its player, into a round hole, modern web design with browsers.” Appx17615 at 1040:19–21; *see also supra* SOC § II.D.2. GD’s counsel made the exact same statement during Opening, which only underscores its significance: “Express Mobile knows this industry shift means it has to fit a square peg, its device-specific player, into a round hole, modern website design, with browsers.” Appx17325 at 114:17–19.

The district court cites *Finisar Corp. v. DirecTV Grp., Inc.*, 523 F.3d 1323 (Fed. Cir. 2008) in support of its finding that any contradiction in the present case was not significant enough to warrant JMOL or a new trial. Appx130. But in *Finisar*, the correct construction was markedly different than the construction used

by the jury in, compelling the Federal Circuit to vacate the jury's verdict. 523 F.3d at 1333. Likewise, in the present case, there is a significant disparity between the court's construction and GD's improper construction that excludes "platform" upon which GD's non-infringement argument relies.

In addition, GD's minimal testimony and slides that the district court highlights as consistent with its claim construction do not save the otherwise improper contradictions. Appx129. Specifically, at 618:12–19:23, 697:23–698:10, and 699:24–700:8, Mr. Kent generically states that he "t[ook] into account" the court's claim constructions and recites the constructions. Appx17452; Appx17472. Simply saying that he is applying the court's claim constructions and parroting those constructions on slides does not make it so. Mr. Kent clearly went on to testify that the court's construction was limited to operating systems and excluded platforms. *E.g.*, Appx17472 at 698:18–699:1 (confirming opinion that a "web browser" is not within "definition" of claim construction). The district court also curiously cites a passage where Mr. Kent is asked on cross to confirm his opinion that "the claimed player in this case is limited to a device dependent player, one for each physical device." Appx17466 at 676:16–23. In the testimony that followed, which the district court did not cite, Mr. Kent continued on to confirm that it was his opinion that the claimed device-dependent Player was indeed limited to one for each device,

which is not applying the proper construction. Appx17466–17467 at 676:24–677:17.

## **2. The District Court Erred In Finding There Were Alternative Bases For Non-Infringement.**

The district court further erred in dismissing GD’s improper claim construction contradictions relating to “Player.” Appx131. At trial, GD presented that the accused instrumentalities lacked “(1) ‘Player’ code, (2) ‘symbolic names,’ or (3) a ‘registry.’” Appx132. The district court, in entirely conclusory manner, determined that it “presumes . . . that the jury credited” GD’s expert testimony over XMO’s as to those three non-infringement bases. *Id.* That conclusory determination was made in error and does not meet applicable JMOL standards. What the district court failed to appreciate was that the issues with the claim construction of the “symbolic names” and the “registry” were similar to those associated with the “Player” term.

The asserted claims of the ’755 and ’287 patents also recite a registry that includes symbolic names, and the ’044 patent recites a “computer memory storing” symbolic names. Appx382 at 37:7–13; Appx423 at 37:50–61; Appx463 at 37:48–61.<sup>7</sup> “Registry” was construed to mean “a database that is used for computing

---

<sup>7</sup> Accordingly, the “registry” non-infringement position acknowledged by the district court is not at issue for the ’044 patent, which thus presents a stronger basis for reversal or vacatur.

functionality.” Appx027. Yet again, XMO seeks to enforce the full scope of the district court’s claim construction, whereas GD seeks to narrowly limit that construction to only particular databases.

As to “registry”, the district court summarized the dispute as GD seeking a “structured database” whereas XMO was seeking a finding that “database . . . can include a file and need not be structured.” Appx128. Ultimately, the district court found, applying the same reasons as the Player term, that GD did not contradict its construction and that any narrowed presentation was not significant enough to warrant JMOL. Appx128–130. The district court erred for the same reasons it did so with respect to the “Player” term.

The district court’s own opinion as to the “registry” term, makes its error self-evident: “On the ‘registry’ term, contrary to Express Mobile’s view, the Court’s construction does not *require* it to include a simple, unstructured database.” Appx129. Indeed, GD and its expert based their non-infringement positions on an erroneously narrow claim scope inconsistent with the court’s actual construction, testifying and arguing that the “database” must be a “structured database.” Appx17460 at 650:2–651:4 (Mr. Kent testifying that “database . . . has to be structured”); Appx17617 at 1050:17–20 (GD counsel arguing that accused functionality is not “a structured database”). But the court’s construction is “*a database* that is used for computing functionality.” Appx027. The district court had

it the wrong way around. A database is a database. There was no basis for refusing to apply the scope of the term as construed. Thus, the district court erred in denying XMO's attempt to enforce the full scope of the claimed registry, which is not limited to any particular database, and in permitting GoDaddy's contradictory claim construction to withstand scrutiny.

As to "symbolic names," GD's purported third and final basis for non-infringement, the district court did not address XMO's JMOL arguments in its order. Specifically, GD's "symbolic names" non-infringement positions were strawman arguments addressing YouTube and other features that were not the "symbolic names" accused by XMO at trial. Appx21696–21697. GD provided no response in its opposition and the district court did not find that a proper non-infringement defense on "symbolic names" was presented. Instead, GD exclusively argued that YouTube functionalities were not symbolic names. Appx21740–21741. Moreover, "symbolic names" is part of the "registry" term and is thus plagued with the same error addressed *supra*. GD admitted as much in its JMOL opposition, stating that "the existence of a 'registry' is required for the 'symbolic name' claim element to be infringed, and vice versa." Appx21739.

Accordingly, the district court erred in finding that there were alternative bases for non-infringement.

### **3. The District Court Erred In Denying A New Trial.**

During the trial, the jury returned a general verdict that the accused products did not infringe the asserted claims. However, in light of the erroneous and contradictory claim construction arguments made by GD at trial concerning at least the term “Player,” it remains impossible to discern which specific limitations the jury found were not met by the accused product. In such instances, this Court has consistently ordered a new trial.

For example, in *Network-1 Technologies*, the district court correctly construed one disputed claim limitation, “low level current,” but erred in its construction of another term, “main power source.” 981 F.3d at 1025–26. This Court determined that the general verdict of non-infringement made it impossible to ascertain which specific limitations the jury found were not met by the accused product, and the erroneous claim construction related to the “main power source” term prejudiced the patentee’s case. *Id.* at 1025. As a result, the Federal Circuit concluded that a new trial on the issue of infringement was necessary. *Id.* Similarly, here, XMO faces a general verdict led by GD’s incorrect claim construction, making it impossible to determine which specific limitations the jury found were not met by the accused products.

This Court has consistently ordered a new trial when unresolved claim construction disputes had an effect on the verdict. *See, e.g., Omega Patents, LLC v.*

*CalAmp Corp.*, 920 F.3d 1337, 1347 (Fed. Cir. 2019) (ordering a new trial and explaining that even if the specification defined the claim term and the parties had agreed on a construction, the district court still had a duty to provide a construction to the jury); *Avid Tech.*, 812 F.3d at 1042 (remanding for a new trial on the issue of infringement where accused infringer had not offered non-infringement under the proper claim construction).

Here, in denying XMO's motion for a new trial, Judge Kennelly found that arguments made by GD "as a whole" were not "so constantly and effectively addressed to the prejudices of the jury that [the Court] must order a new trial." Appx132 (citing *Draper v. Airco, Inc.*, 580 F.2d 91, 97 (3d Cir. 1978)). But as established *supra*, GD's counsel and experts consistently presented erroneous and contradictory claim construction arguments that were constantly aimed at misleading the jury. Those repeated and extreme statements throughout trial are much more pervasive than those at issue in *Draper*, where prejudicial statements were included during closing. *See Draper*, 580 F.2d at 97. This Court should order a new trial to correct the error that pervaded the trial through GD's contradictory claim construction evidence and argument. *See Network-1 Technologies, Inc.*, 981 F.3d at 1025–26; *Omega Patents, LLC*, 920 F.3d at 1347; *Avid Tech.*, 812 F.3d at 1042.



## CONCLUSION AND STATEMENT OF RELIEF SOUGHT

The district court erred in granting summary judgment of non-infringement as to the '397 patent family, including an erroneous construction of “runtime engine” and an erroneous application of that construction even if accepted. XMO respectfully requests reversal of summary judgment of non-infringement. The district court also erred in denying JMOL and/or a new trial as to the '755 patent family, including an erroneous refusal to find that GD contradicted its construction of “Player” and “device-dependent code.” XMO respectfully requests reversal or vacatur of the denial of JMOL, and alternatively, a new trial.

November 13, 2023

/s/ James R. Nuttall

James R. Nuttall  
Katherine H. Tellez  
Robert F. Kappers  
Candice J. Kwark  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300  
Facsimile: (312) 577-1370

*Counsel for Plaintiff-Appellant,  
Express Mobile, Inc.*

## **ADDENDUM**

**ADDENDUM**  
**TABLE OF CONTENTS**

	<b>Page</b>
Memorandum Opinion and Order, dated June 1, 2021 .....	Appx1
Order on Claim Construction dated June 8, 2021 .....	Appx26
Memorandum Opinion and Order dated, August 8, 2022 .....	Appx29
Oral Order, dated October 27, 2022.....	Appx82
Final Jury Instructions, dated March 3, 2023 .....	Appx84
Jury Verdict, dated March 3, 2023.....	Appx112
Judgment after Verdict in a Civil Case, dated March 6, 2023.....	Appx118
Memorandum Opinion and Order, dated July 5, 2023 .....	Appx120
United States Patent No. US 6,546,397 .....	Appx139
United States Patent No. US 7,594,168 .....	Appx243
United States Patent No. US 9,063,755 .....	Appx345
United States Patent No. US 9,471,287 .....	Appx385
United States Patent No. US 9,928,044 .....	Appx425

IN THE UNITED STATES DISTRICT COURT  
FOR THE DISTRICT OF DELAWARE

EXPRESS MOBILE, INC.,

Plaintiff,

v.

GODADDY.COM, LLC,

Defendant.

Civil Action No. 19-1937-RGA

MEMORANDUM OPINION

Timothy Devlin, DEVLIN LAW FIRM LLC, Wilmington, DE; James R. Nuttall, Michael Dockterman, Robert F. Kappers, Tron Fu, Katherine H. Johnson, STEPTOE & JOHNSON LLP, Chicago, IL; Christopher A. Suarez, STEPTOE & JOHNSON LLP, Washington, DC, Attorneys for Plaintiff.

Beth Moskow-Schnoll, Brittany Giusini, Brian S.S. Auerbach, BALLARD SPAHR LLP, Wilmington, DE; Brian W. LaCorte, Jonathan A. Talcott, BALLARD SPAHR LLP, Phoenix, AZ, Attorneys for Defendant.

June 1, 2021

/s/ Richard G. Andrews

**ANDREWS, UNITED STATES DISTRICT JUDGE:**

Before me is the issue of claim construction of multiple terms in U.S. Patent Nos. 6,546,397 (the '397 Patent), 7,594,168 (the '168 Patent), 9,063,755 (the '755 Patent), 9,471,287 (the '287 Patent), and 9,928,044 (the '044 Patent). I have considered the Parties' Joint Claim Construction Brief. (D.I. 64). I held remote oral argument on April 8, 2021. (D.I. 77). The parties argued ten terms there. I asked for supplemental briefing on two of them, which is underway. (D.I. 80). I now decide the other eight terms.

### **I. LEGAL STANDARD**

“It is a bedrock principle of patent law that the claims of a patent define the invention to which the patentee is entitled the right to exclude.” *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312 (Fed. Cir. 2005) (en banc) (internal quotation marks omitted). “[T]here is no magic formula or catechism for conducting claim construction.’ Instead, the court is free to attach the appropriate weight to appropriate sources ‘in light of the statutes and policies that inform patent law.’” *SoftView LLC v. Apple Inc.*, 2013 WL 4758195, at \*1 (D. Del. Sept. 4, 2013) (quoting *Phillips*, 415 F.3d at 1324) (alteration in original). When construing patent claims, a court considers the literal language of the claim, the patent specification, and the prosecution history. *Markman v. Westview Instruments, Inc.*, 52 F.3d 967, 977–80 (Fed. Cir. 1995) (en banc), *aff’d*, 517 U.S. 370 (1996). Of these sources, “the specification is always highly relevant to the claim construction analysis. Usually, it is dispositive; it is the single best guide to the meaning of a disputed term.” *Phillips*, 415 F.3d at 1315 (internal quotation marks omitted).

“[T]he words of a claim are generally given their ordinary and customary meaning. . . . [Which is] the meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention, i.e., as of the effective filing date of the patent application.”

*Id.* at 1312–13 (citations and internal quotation marks omitted). “[T]he ordinary meaning of a claim term is its meaning to [an] ordinary artisan after reading the entire patent.” *Id.* at 1321 (internal quotation marks omitted). “In some cases, the ordinary meaning of claim language as understood by a person of skill in the art may be readily apparent even to lay judges, and claim construction in such cases involves little more than the application of the widely accepted meaning of commonly understood words.” *Id.* at 1314.

When a court relies solely upon the intrinsic evidence—the patent claims, the specification, and the prosecution history—the court’s construction is a determination of law. *See Teva Pharm. USA, Inc. v. Sandoz, Inc.*, 574 U.S. 318, 331 (2015). The court may also make factual findings based upon consideration of extrinsic evidence, which “consists of all evidence external to the patent and prosecution history, including expert and inventor testimony, dictionaries, and learned treatises.” *Phillips*, 415 F.3d at 1317–19 (internal quotation marks omitted). Extrinsic evidence may assist the court in understanding the underlying technology, the meaning of terms to one skilled in the art, and how the invention works. *Id.* Extrinsic evidence, however, is less reliable and less useful in claim construction than the patent and its prosecution history. *Id.*

“A claim construction is persuasive, not because it follows a certain rule, but because it defines terms in the context of the whole patent.” *Renishaw PLC v. Marposs Societa’ per Azioni*, 158 F.3d 1243, 1250 (Fed. Cir. 1998). It follows that “a claim interpretation that would exclude the inventor’s device is rarely the correct interpretation.” *Osram GMBH v. Int’l Trade Comm’n*, 505 F.3d 1351, 1358 (Fed. Cir. 2007) (citation and internal quotation marks omitted).

## II. BACKGROUND

The specifications of the '397 and '168 patents are “substantively identical” (D.I. 71 at 6), as are, separately, the specifications of the '755, '287, and '044 patents (*id.* at 14). The two common specifications are different from each other; for claim construction purposes, the two sets of patents are unrelated and therefore extrinsic evidence to each other. The inventions display an in-work webpage in real time so a web developer can view the webpage during editing as it would appear to an end user viewing the webpage through a browser. The following claims are the most relevant for the purposes of this Markman:

### Claim 1 of the '397 Patent

1. A method to allow users to produce Internet websites on and for computers having a browser and a virtual machine capable of generating displays, said method comprising:
  - (a) presenting a viewable menu having a user selectable panel of settings describing elements on a website, said panel of settings being presented through a browser on a computer adapted to accept one or more of said selectable settings in said panel as inputs therefrom, and where at least one of said user selectable settings in said panel corresponds to commands to said virtual machine;
  - (b) generating a display in accordance with one or more user selected settings substantially contemporaneously with the selection thereof;
  - (c) storing information representative of said one or more user selected settings in a database;
  - (d) generating a website at least in part by retrieving said information representative of said one or more user selected settings stored in said database; and
  - (e) building one or more web pages to generate said website from at least a portion of said database and *at least one run time file*, where said *at least one run time file* utilizes information stored in said database to generate virtual machine commands for the display of at least a portion of said one or more web pages.

(D.I. 1-1, Ex. A (“the '397 patent”), claim 1) (emphasis added).

### Claim 2 of the '397 Patent

2. An apparatus for producing Internet websites on and for computers having a browser and a virtual machine capable of generating displays, said apparatus comprising:
- (a) an interface to present a viewable menu of a user selectable panel of settings to describe elements on a website, said panel of settings being presented through a browser on a computer adapted to accept one or more of said selectable settings in said panel as inputs therefrom, and where at least one of said user selectable settings in said panel corresponds to commands to said virtual machine;
  - (b) a browser to generate a display in accordance with one or more user selected settings substantially contemporaneously with the selection thereof;
  - (c) a database for storing information representative of said one or more user selected settings; and
  - (d) a build tool having at least one run time file for generating one or more web pages, said run time file operating to utilize information stored in said database to generate commands to said virtual machine for generating the display of at least a portion of said one or more web pages.

(*Id.*, claim 2).

**Claim 9 of the '397 Patent**

9. The apparatus of claim 2, wherein said elements include a button or an images (sic), wherein said selectable settings includes the selection of an element style, and wherein said *build engine* includes means for storing information representative of selected style in said database.

(*Id.*, claim 9) (emphasis added).

**Claim 1 of the '168 Patent**

1. A system for assembling a web site comprising:

a server comprising a *build engine* configured to:

accept user input to create a web site, the web site comprising a plurality of web pages, each web page comprising a plurality of objects.

accept user input to associate a style with objects of the plurality of web pages, wherein each web page comprises at least one button object or at least one image object, and wherein the at least one button object or at least one image object is associated with a style that includes values defining transformations and time lines for the at least one button object or at least one image object; and wherein



each web page is defined entirely by each of the plurality of objects comprising the web page and the style associated with the object,

produce a database with a multidimensional array comprising the objects that comprise the web site including data defining, for each object, the object style, an object number, and an indication of the web page that each object is part of, and

provide the database to a server accessible to web browser;

wherein the database is produced such that a web browser with access to a *runtime engine* is configured to generate the web-site from the objects and style data extracted from the provided database.

(D.I. 1-2, Ex. B. (“the ’168 patent”), claim 1) (emphasis added).

#### **Claim 1 of the ’755 Patent**

1. A system for generating code to provide content on a display of a device, said system comprising:

computer memory storing a *registry* of:

(a) symbolic names required for evoking one or more *web components* each related to a set of inputs and outputs of a *web service* obtainable over a network, where the *symbolic names* are character strings that do not contain either a persistent address or pointer to an output value accessible to the *web service*, and

(b) the address of the *web service*;

an authoring tool configured to:

define a user interface (UI) object for presentation on the display, where said UI object corresponds to the *web component* included in said *registry* selected from the group consisting of an input of the web service and an output of the web service,

access said computer memory to select the symbolic name corresponding to the *web component* of the defined UI object,

associate the selected *symbolic name* with the defined UI object,

produce an *Application* including the selected *symbolic name* of the defined UI object, where said *Application* is a device-independent code, and

produce a Player, where said Player is a device-dependent code;

such that, when the *Application* and Player are provided to the device and executed on the device, and when a user of the device provides one or more input values associated with an input *symbolic name* to an input of the defined UI object,

- 1) the device provides the user provided one or more input values and corresponding input *symbolic name* to the web service,
- 2) the *web service* utilizes the input *symbolic name* and the user provided one or more input values for generating one or more output values having an associated output *symbolic name*,
- 3) said Player receives the output *symbolic name* and corresponding one or more output values and provides instructions for a display of the device to present an output value in the defined UI object.

(D.I. 1-3, Ex. C (“the ’755 patent”), claim 1) (emphasis added).

#### **Claim 1 of the ’287 Patent**

1. A system for generating code to provide content on a display of a device, said system comprising:

computer memory storing a *registry* of:

- (c) *symbolic names* required for evoking one or more *web components* each related to a set of inputs and outputs of a *web service* obtainable over a network, where the *symbolic names* are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, where each *symbolic name* has an associated data format class type corresponding to a subclass of User Interface (UI) objects that support the data format type of the *symbolic name*, and has a preferred UI object, and
- (d) an address of the *web service*;

an authoring tool configured to:

define a (UI) object for presentation on the display, where said defined UI object corresponds to a *web component* included in said *registry* selected from a group consisting of an input of the *web service* and an output of the *web service*, where each defined UI object is either: 1) selected by a user of the authoring tool; or 2) automatically selected by the system as the preferred UI object corresponding to the *symbolic name* of the *web component* selected by the user of the authoring tool,

access said computer memory to select the *symbolic name* corresponding to the *web component* of the defined UI object,

associate the selected *symbolic name* with the defined UI object, where the selected *symbolic name* is only available to UI objects that support the defined data format associated with that *symbolic name*, and

produce an *Application* including the selected *symbolic name* of the defined UI object, where said *Application* is a device-independent code; and

a Player, where said Player is a device-dependent code, wherein, when the *Application* and Player are provided to the device and executed on the device, and when the user of the device provides one or more input values associated with an input *symbolic name* to an input of the defined UI object,

- 1) the device provides the user provided one or more input values and corresponding input *symbolic name* to the *web service*,
- 2) the *web service* utilizes the input *symbolic name* and the user provided one or more input values for generating one or more output values having an associated output *symbolic name*,
- 3) said Player receives the output *symbolic name* and corresponding one or more output values and provides instructions for the display of the device to present an output value in the defined UI object.

(D.I. 1-4, Ex. D (“the ’287 patent”), claim 1) (emphasis added).

#### **Claim 1 of the ’044 Patent**

1. A system for generating code to provide content on a display of a device, said system comprising:

computer memory storing:

- (e) *symbolic names* required for evoking one or more *web components* each related to a set of inputs and outputs of a *web service* obtainable over a network, where the *symbolic names* are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, where each *symbolic name* has an associated data format class type corresponding to a subclass of User Interface (UI) objects that support the data format type of the *symbolic name*, and where each *symbolic name* has a preferred UI object, and
- (f) an address of the *web service*;

an authoring tool configured to:

define a (UI) object for presentation on the display,

where said defined UI object corresponds to a *web component* included in said computer memory selected from a group consisting of an input of the *web service* and an output of the *web service*, where each defined UI object is either:

- 1) selected by a user of the authoring tool; or
- 2) automatically selected by the system as the preferred UI object corresponding to the *symbolic name* of the *web component* selected by the user of the authoring tool,

access said computer memory to select the *symbolic name* corresponding to the *web component* of the defined UI object,

associate the selected *symbolic name* with the defined UI object, where the selected *symbolic name* is only available to UI objects that support the defined data format associated with that *symbolic name*,

store information representative of said defined UI object and related settings in a database;

retrieve said information representative of said one or more said UI object settings stored in said database; and

build an *application* consisting of one or more web page views from at least a portion of said database utilizing at least one player, where said player utilizes information stored in said database to generate for the display of at least a portion of said one or more web pages,

wherein when the *application* and player are provided to the device and executed on the device, and

when the user of the device provides one or more input values associated with an input *symbolic name* to an input of the defined UI object, the device provides the user provided one or more input values and corresponding input *symbolic name* to the *web service*, the *web service* utilizes the input *symbolic name* and the user provided one or more input values for generating one or more output values having an associated output *symbolic name*,

and the player receives the output *symbolic name* and corresponding one or more output values and provides instructions for the display of the device to present an output value in the defined UI object.

(D.I. 1-5, Ex. E (“the ’044 patent”), claim 1) (emphasis added).

**III. CONSTRUCTION OF AGREED-UPON TERMS**

I adopt the following agreed-upon constructions:

<b>Claim Term</b>	<b>Construction</b>
multi-dimensional array(s) / multidimensional array(s)	“a uniquely identifiable indexed set of related elements, wherein each element is addressed by a set of two or more indices, each index corresponding to a dimension of the array”
storing information representative of said one or more user selected setting in a database	“storing data in a database, which data pertains to one or more attributes of an object available for selection by a user”
Transformation	“the changing of an object from one state to another based on a timer control, subject to user settings”
Settings	“attributes of an object available for selection”
authoring tool / authoring tool configured to	a system, with a graphical interface, for generating code to display content on a device screen
device-dependent code	code that is specific to the operating system, programming language, or platform of a device
device-independent code	code that is not specific to the operating system, programming language, or platform of a device
where said application is device-dependent code	where said Application is a device-independent code <sup>1</sup>
for evoking one or more web components	for calling up one or more web components

**IV. CONSTRUCTION OF DISPUTED TERMS**

I have construed below the terms the parties selected for oral argument. The parties may choose to argue claim construction issues for other terms as necessary to any summary judgment

<sup>1</sup> I am not positive the parties meant to capitalize the “a” in Application. But since they did, I do too.

motions, but, especially given the sparse briefing on many of the disputed terms, I do not decide them here.

**1. “at least one run time file / one or more run time files” (’397/1, 2, 37)**

- a. *Plaintiff’s proposed construction:*
  - i. “one or more files, including a run time engine, that are downloaded or created when a browser is pointed to a web page or website”
- b. *Defendant’s proposed construction:*
  - i. “one or more files, including a run time engine, that are downloaded or created and executed by a browser when a browser is pointed to a web page or website”
- c. *Court’s construction:*
  - i. “one or more files, including a run time engine, that are downloaded or created when a browser is pointed to a web page or website”

The parties dispute whether the “run time file(s)” must be executable by a browser.

Plaintiff argues that though some “run time files” may be executable, other files downloaded by a browser to display web page content, such as image, audio, or video files, are not executable. (D.I. 64 at 31). Plaintiff further asserts that specifying “executable” is unnecessary because the run time engine file is by its nature executable. (*Id.*).

Defendant agrees that the run time engine file is necessarily executable. (*Id.*). It argues that, because the “files” can, as Plaintiff asserts, include non-executable files, it is important to explicitly require that at least one file be executable. (*Id.* at 32).

I agree with Plaintiff. The construction requires a run time engine file. If a run time engine file is executable, which both parties agree is the case, there is no meaningful difference between the two constructions other than that Defendant’s inclusion of “executed” is redundant. I therefore construe “at least one run time file / one or more run time files” as “one or more files, including a run time engine, that are downloaded or created when a browser is pointed to a web page or website.”

2. “build engine” (’397/9, 14, 19, 23; ’168/1, 2)

- a. *Plaintiff’s proposed constructions:*
  - i. no construction necessary; or
  - ii. “a software component for processing user input related to building a website/webpage(s) for database storage”
- b. *Defendant’s proposed constructions:*
  - i. ’397 patent:
    1. indefinite; or
    2. same construction as for ’168 patent
  - ii. ’168 patent
    1. “component that receives data or information regarding the creation, editing and/or display of a web page and updates one or more databases, including a database internal to the build engine, in response to that information or data”
- c. *Court’s construction:*
  - i. not indefinite; “build engine” means “build tool” in the ’397 patent; any construction of “build engine” in the ’168 patent is deferred

Defendant argues that “build engine” in the ’397 patent is indefinite because it lacks antecedent basis, so that a POSA would not understand what “build engine” refers to with reasonable certainty. (D.I. 64 at 23–24). Although Defendant acknowledges the USPTO’s Certificate of Correction (COC), issued to correct “build engine” to “build tool” in the ’397 patent (D.I. 65-9, Ex. 1I at 2), it presents two arguments against the use of the COC for claim construction of this term. First, Defendant argues that because the COC was not issued until February 13, 2018 (*id.*), to the extent the COC is applicable, it does not apply before its date of issuance (D.I. 77 at 32). Second, Defendant argues that, in any case, the COC should be disregarded because the specification, when discussing Fig. 3A, illustrates that a “build engine” is a “build tool” component. (D.I. 64 at 24, 27). The two terms therefore cannot be equivalent, Defendant maintains. (*Id.* at 27).

Plaintiff, on the other hand, argues that no construction is necessary because the COC makes clear that the antecedent basis for “build engine” is “build tool.” (*Id.* at 22–23). Plaintiff also argues that no construction is necessary because the specification equates the two terms

when it states the phrase “build engine (i.e. build tool).” (*Id.* at 22) (citing ’397 patent at 2:1–4). For the same reason, Plaintiff maintains, the COC properly treats “build engine” and “build tool” as interchangeable terms. (*Id.* at 26).

There are therefore two issues related to the COC. First, whether the COC’s corrections only apply after the COC’s date of issuance. Second, whether the COC’s correction is proper.

The Federal Circuit has held that “for causes arising after the PTO issues a certificate of correction, the certificate of correction is to be treated as part of the original patent—i.e., as if the certificate had been issued along with the original patent.” *Southwest Software, Inc. v. Harlequin, Inc.*, 226 F.3d 1280, 1295 (Fed. Cir. 2000). For causes of action that arise after the date of the COC’s issuance, therefore, the COC’s corrected claim language has effect dating back to the priority date of the patent because “the certificate is considered part of the original patent.” *Id.*

The COC was issued on February 13, 2018. (D.I. 65-9, Ex. 1I at 2). Plaintiff alleges Defendant was made aware of its infringement as early as February 28, 2013 but does not specify the date of initial infringement. (D.I. 47 at 16, 25). Relevant to the issue here, Plaintiff alleges infringement before and after the date of the COC’s issuance. Because the COC’s corrected language is treated as “part of the original patent,” for any alleged infringement after the date of the COC’s issuance, February 13, 2018, the corrected claim language applies.

The corrected claim language does not apply, on the other hand, to alleged infringement prior to February 13, 2018. If a COC were given effect as of the priority date of the patent for infringement *prior* to the date of the COC’s issuance, a party could be held liable for infringing a facially invalid patent. “In such a case, where the claim is invalid on its face without the certificate of correction, it strikes us as an illogical result to allow the patent holder, once the certificate of correction has issued, to sue an alleged infringer for activities that occurred before



the issuance of the certificate of correction.” *Southwest Software*, 226 F.3d at 1295–96. The corrected claim language therefore does not apply to any alleged infringement prior to February 13, 2018, the date of COC issuance.

The second issue relates to whether the COC’s issuance was proper.

Certificates of correction may be issued for “a mistake of a clerical or typographical nature, or of minor character” at the discretion of the USPTO. 35 U.S.C. § 255. The Federal Circuit has held that although § 255 does “allow broadening corrections of clerical or typographical mistakes,” there is clear Congressional intent “to protect the public against the unanticipated broadening of a claim after the grant of the patent by the PTO.” *Superior Fireplace Co. v. Majestic Products Co.*, 270 F.3d 1358, 1371 (Fed. Cir. 2001). “It would be inconsistent with that objective to interpret § 255 to allow a patentee to broaden a claim due to the correction of a clerical or typographical mistake that the public could not discern from the public file and for which the public therefore had no effective notice.” *Id.* The “public file” used to determine whether a correction is proper consists of “the specification, drawings, and prosecution history.” *Id.* at 1372. As for a mistake of “minor character”: “A mistake that, if corrected, would broaden the scope of a claim must thus be viewed as highly important and thus cannot be a mistake of ‘minor character.’” *Id.* at 1375.

In the ’397 patent, the relevant claim recites, “The apparatus of claim 2, wherein said elements include a button or an images (sic), wherein said selectable settings includes the selection of an element style, and wherein said build engine includes means for storing information representative of selected style in said database.” (’397 patent at 66:48–49 (uncorrected claim 9)). There is no grammatical error in the use of “build engine” that suggests a mistake on its face. Looking to independent claim 2, however, although dependent claim 9

references “said build engine,” claim 2 recites only “a build tool,” not a “build engine.” This raises the possibility of a clerical or typographical error in the use of “said build engine.”

The specification teaches that a “build engine” is an example of a “build tool.” (’397 patent at 9:1–29). Although Defendant argues that these two terms are distinguished in Fig. 3A, the relevant specification language stating that a “build engine” is an example of a “build tool” explicitly refers to Fig. 3A. (*Id.*). Looking at the claim language, claim 9 references “said build engine” while claim 2 recites a “build tool.” Because the specification states that a “build engine” is an example of a “build tool,” it is therefore reasonable to expect that a reader of the patent could discern that “build engine” in claim 9 of the ’397 patent is meant to refer to the “build tool” in claim 2.

I therefore find that the COC’s correction of “build engine” to “build tool” in the ’397 patent is a valid correction of a “clerical or typographical mistake” under § 255.

For the construction of “build engine” in the ’397 patent as it applies to infringement prior to the COC’s date of issuance, I analyze indefiniteness. “[A] patent is invalid for indefiniteness if its claims, read in light of the specification delineating the patent, and the prosecution history, fail to inform, with reasonable certainty, those skilled in the art about the scope of the invention.” *Nautilus, Inc. v. Biosig Instruments, Inc.*, 572 U.S. 898, 901 (2014). In order to prove a patent is invalid for indefiniteness, there must be clear and convincing evidence. *Id.* at 912 n.10 (citing *Microsoft Corp. v. i4i Ltd. Partnership*, 564 U.S. 91, 95 (2011)).

I do not think Defendant has proven indefiniteness by clear and convincing evidence for many of the same reasons cited in the COC analysis. The specification states that a “build engine” is an example of a “build tool,” specifically in relation to Fig. 3A (’397 patent at 9:1–29), which Defendant argues distinguishes the two (D.I. 64 at 27). No other argument was

presented in the briefing or at oral argument that a POSA would not have been able to determine, with reasonable certainty, that the “build engine” in claim 9 referred to the “build tool” in claim

2. Because the specification notes that a “build engine” is an example of a “build tool,” I think a POSA would have understood with reasonable certainty that “said build engine” in claim 9 referred to the “build tool” in claim 2.

I therefore find for the ‘397 patent that “build engine” is not indefinite, and that “build engine” should be construed as “build tool.” Because the parties did not present “build tool” for construction at oral argument, I will not construe “build tool” here. The parties may present any claim construction disputes regarding “build tool” in their summary judgment briefing.<sup>2</sup>

### 3. **“run time (runtime) engine” (’168/1)**

- a. *Plaintiff’s proposed constructions:*
  - i. “file that is executed at runtime that utilizes information from the database and generates commands to display a web page or website”; or
  - ii. “file that is executed at runtime that reads information from the database and generates commands to display a web page or website”
- b. *Defendant’s proposed construction:*
  - i. “file that is executed at runtime that reads information from the database and generates virtual machine commands to display a web page or website”
- c. *Court’s construction:*
  - i. “file that is executed at runtime that reads information from the database and generates commands to display a web page or website”

I previously construed “runtime engine” to mean “file that is executed at runtime that reads information from the database and generates commands to display a web page or website.” (D.I. 65-6, Ex. 1E at 5). To the extent Plaintiff disagrees with this construction, it argues that “utilizes information” could mean something more than simply reading information from the

---

<sup>2</sup> I also do not construe “build engine” in the ‘168 patent. The proposed construction was barely briefed (D.I. 64 at 22-27) and was barely mentioned at the Markman hearing. (D.I. 77 at 36, 39-40).

database. (D.I. 64 at 38–39). At oral argument, Plaintiff focused its argument on concerns that Defendant may later argue that reading and downloading information should be construed differently. (D.I. 77 at 49:21–50:21). Defendant agreed that “downloading is equivalent to reading over a network.” (*Id.* at 46:14–47:3). I therefore focus my analysis on the difference between my prior construction, which Plaintiff accepted as an alternative construction, and Defendant’s construction, which construes “commands” as limited solely to “virtual machine commands.”

Plaintiff argues that the claim language does not refer explicitly to virtual machine commands. (D.I. 64 at 40). Plaintiff also maintains that the specification “confirms that a run time engine can generate other types of commands to display a web page or website, such as commands for downloading image, audio and video files.” (*Id.*) (citing ’168 patent at 5:49–68). Defendant, on the other hand, argues that the runtime engine “generates the web-site” using information stored in the database, a process for which the specification describes issuing virtual machine commands. (*Id.* at 42).

I agree with Plaintiff. The term “virtual machine” does not appear in the claim language. Defendant’s citation to the specification for the process of generating the web-site is only in reference to an embodiment; it does not indicate that the process must exclusively be done using virtual machine commands. (*See* ’168 patent at 35:8–12). The specification does not otherwise limit the commands involved in displaying the web page to virtual machine commands. I therefore construe “runtime engine” as “file that is executed at runtime that reads information from the database and generates commands to display a web page or website.”

4. **“registry” (’755/1, 12; ’287/1, 15)**

- a. *Plaintiff’s proposed constructions:*
  - i. no construction necessary; or

- ii. “a database or lookup table that is used for computing functionality”
- b. *Defendant’s proposed construction*:
  - i. “a database, XML file, or Portable Description Language file that exists on a computer”
- c. *Court’s construction*:
  - i. “a database that is used for computing functionality”

The parties generally agree that “registry” is a term of art in the computer science and web design fields. (D.I. 64 at 45–46). They dispute whether Plaintiff acted as its own lexicographer to limit the definition of “registry” to that provided in the specification. (*Id.*).

Plaintiff argues that “registry” should not be limited to “a database, XML file, or Portable Description Language” because that description, while in the specification, refers only to “one embodiment that references the ‘web component registry.’” (*Id.* at 45) (citing ’755 patent at 8:19–22). Other embodiments, Plaintiff argues, disclose a web component registry that could be implemented in ways other than “database, XML file, or Portable Description Language,” such as by using a list with relevant information. (*See id.* at 46) (citing ’755 patent at 9:16–26, 22:26–29).

Defendant, on the other hand, argues that the specification reference to “web component registry 230” is definitional because it does not contain language limiting the term to a single embodiment. (D.I. 64 at 45).

I generally agree with Plaintiff. The language in the specification referring to “web component registry” is limited to the description of Fig. 2A, which is described as “an embodiment.” (’755 patent at 7:63–66). Nothing in the specification indicates the patentee’s desire to redefine or limit the term “registry” to only a “database, XML file, or Portable Description Language”; that description is merely used to describe one such manifestation of the element in an embodiment.

I do not, however, agree with Plaintiff's inclusion of a "lookup table" in its construction. Plaintiff cites to the specification to support its assertion that "registry" can take a tabular format, but the cited portions only teach a list with information, not a table. ('755 patent at 9:16–26, 22:26–29). The parties do at minimum agree, though, that the construction of "registry" requires a "database."

It might be helpful to a jury to have a construction. I will adopt Plaintiff's alternative construction, except for "lookup table," and I therefore construe "registry" as "a database that is used for computing functionality."

**5. "web component" ('755/1, 3, 6, 7, 12, 14, 17, 18; '287/1, 3, 6, 7, 15, 17, 20, 21; '044/1, 3, 6, 7, 15, 17, 20, 21)**

- a. *Plaintiff's proposed construction:*
  - i. "one or more functionalities associated with one or more web page elements to be displayed on a device"
- b. *Defendant's proposed construction:*
  - i. "software object, which has a clearly defined interface, conforms to a prescribed behavior common to all components within an architecture, is meant to interact with other components, and encapsulates certain functionality or a set of functionalities"
- c. *Court's construction:*
  - i. "software objects that provide functionalities of a web service"

The parties partially agreed at oral argument to construe "web component" as "software objects that provide functionalities of a web service," disputing only whether to include "certain" in front of "functionalities." (D.I. 77 at 70:22–73:14).

Neither party meaningfully argues why "certain" should or should not be included. The scope of "functionalities" provided by the "software objects" is already limited to the context of a "web service." Further specifying only "certain" functionalities does not change the scope of the construction.

I therefore construe “web component” as “software objects that provide functionalities of a web service.”

6. “web service” (’755/1, 5, 7, 12, 16, 18; ’287/1, 5, 7, 15, 19, 21; ’044/1, 5, 7, 15, 19, 21)

- a. *Plaintiff’s proposed constructions:*
  - i. no construction necessary; or
  - ii. “A software system that supports interaction between devices over a network”
- b. *Defendant’s proposed construction:*
  - i. “A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format called “WSDL” (web service description language). Other systems interact with the web service in a manner prescribed by its description using Simple Object Access Protocol (“SOAP”) messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards.”
- c. *Court’s construction:*
  - i. “A software system that supports interaction between devices over a network.”

Plaintiff argues that no construction of “web service” is necessary because it is a well understood term of art for which definitions “abound in computer literature and technical dictionaries.” (D.I. 64 at 50). To the extent a construction is required, Plaintiff asserts, its alternative construction is supported by the specification description of a “web service” as “a plurality of services obtainable over the Internet.” (*Id.*) (citing ’755 patent at 8:18–19).

Defendant instead asserts that “web service” should be construed and that, because during prosecution the application referenced the “World Wide Web” in conjunction with “web service,” the World Wide Web Consortium (W3C) definition should apply here. (D.I. 64 at 50–51). Defendant then asserted at oral argument—without having included any prior reference in its three paragraphs of briefing on this term—that the patentee used the W3C definition as part of an amendment to overcome the prior art McCain reference, which Defendant argued would constitute prosecution history disclaimer. (D.I. 77 at 60:4–24).

Plaintiff replied that the reference to the prosecution history was just general background and not asserted by the patentee in a way that would constitute disclaimer. (*Id.* at 63:23–64:3). Plaintiff also argued that the W3C definition is overly limiting because it excludes web service interfaces, such as REST, that are specifically referenced in the specification. (*Id.* at 56:3–15; *see* ’755 patent at 28:14–18).

I mostly agree with Plaintiff. The reference to the W3C definition in the prosecution history cited by Defendant appears in a section that generally describes the various components of the application, including a “web service.” (D.I. 42-15, Reply to Office Action of April 1, 2012,<sup>3</sup> at 5–6). To the extent that the patentee used this information to overcome the combination of prior art references McCain and Sidman, the patentee did not delimit the type of components constituting the “web service” or use the latter, restrictive portion of the W3C definition in patentee’s assertions. (*Id.*).

As for disclaimer, the parties have not provided sufficient argument for me to evaluate this issue. Defendant’s argument is essentially that the application, during prosecution, referenced the World Wide Web when referring to “web service,” and therefore I should adopt the W3C definition of the term. (D.I. 64 at 50–51). I have already noted that this definition appears in a background section and, consequently, does not constitute disclaimer without other indicia of disavowal. Defendant presents no additional arguments for disclaimer. In fact, the collective briefing on this term covers only two and a half pages, with only a few sentences dedicated to arguing about disclaimer. Prosecution history disclaimer “requires that the alleged disavowing actions or statements made during prosecution be both clear and unmistakable.” *Omega Eng’g, Inc. v. Raytek Corp.*, 334 F.3d 1314, 1325–26 (Fed. Cir. 2003). Based on the

---

<sup>3</sup> The patentee used “Reply to Office Action of April 1, 2012” in the header of the document, but in the text describes a response to an office action dated April 1, 2013 (D.I. 42-15 at 1), which I think is correct.



briefing, it is not apparent to me that the patentee's statements arise to "clear and unmistakable" disclaimer.

Plaintiff's alternative construction better fits the description of "web service" in the specification as a "a plurality of services obtainable over the Internet." ('755 patent at 8:18–19). I therefore construe "web services" as "A software system that supports interaction between devices over a network."

**7. "symbolic name(s)" ('755/1, 12; '287/1, 15; '044/1, 15)**

- a. *Plaintiff's proposed construction:*
  - i. no construction necessary
- b. *Defendant's proposed construction:*
  - i. "parameters specifying inputs and/or outputs associated with web services"
- c. *Court's construction:*
  - i. no construction necessary

Plaintiff argues that no construction is necessary because the claim language itself defines "symbolic names" as "character strings that do not contain either a persistent address or pointer to an output value accessible to the web service." (D.I. 64 at 52) (citing '755 patent, claim 1). Defendant, on the other hand, argues that this definition merely describes what "symbolic names" cannot be and is therefore insufficient. (*Id.* at 54). Defendant also asserts that "symbolic names" should be limited to "parameters" that specify "inputs and/or outputs associated with web services," consistent with patentee's representations to overcome prior art references during prosecution. (*Id.* at 53) (citing D.I. 42-15, Reply to Office Action of April 1, 2012, at 5–6).

I agree with Defendant that the patentee argued that "symbolic names" are parameters associated with inputs or outputs related to web services. But Defendant has not argued how or why this characterization of "symbolic names" subsumes or contradicts the definition set forth in claim 1 of the '755 patent. Nor has Defendant explained why it thinks the definition of

“symbolic names” set forth in claim 1 is impermissibly broader than the characterization of “symbolic names” offered by the patentee during prosecution. Claim 1 affirmatively describes “symbolic names” as “character strings.” The claim further limits “symbolic names” to only those “character strings” that “do not contain either a persistent address or pointer to an output value accessible to the web service.” Not having a “persistent address” was, in fact, the exact characteristic of “symbolic name” asserted by the patentee to overcome prior art references McCain and Sidman during prosecution. (D.I. 42-15, Reply to Office Action of April 1, 2012, at 5–6). Because the language of the claim expressly limits the scope of “symbolic name” in a manner consistent with patentee’s arguments during prosecution—and because Defendant has not otherwise provided reasonable basis for changing the scope based on the prosecution history—there is no need to further construe the term.

No construction of “symbolic names” is necessary.

8. **“application”** (’755/1, 12, 22; ’287/1, 15, 25; ’044/1, 15, 25)

- a. *Plaintiff’s proposed construction:*
  - i. “device-independent software code containing instructions for a device”
- b. *Defendant’s proposed constructions:*
  - i. “device-independent code which contains instructions for a device and which is separate and independent from the Player”; or
  - ii. “device-independent code which contains instruction for a device and which is separate and distinct from the Player”
- c. *Court’s construction:*
  - i. “device-independent code containing instructions for a device and which is separate from the Player”

At oral argument, Plaintiff preserved its objection to incorporating “separate” in the construction, but otherwise agreed that the application is separate from the player. (D.I. 77 at 74:17–75:3). Plaintiff also agreed that the word “software” could be taken out of its construction

without changing the meaning. (*Id.* at 74:12–15). I therefore focus my analysis on whether the construction should read “separate *and independent*” or just “separate.”

Plaintiff argues that the specification and prosecution history do not use the “separate and independent” language to delimit the scope of “application.” (D.I. 77 at 75:17–20). Plaintiff also asserts that although I included “separate” in my prior construction of “application” (i.e. “device-independent code which contains instructions for a device and which is separate from the Player”), I rejected the requirement that the “application” be “partitioned” from the “Player.” (*Id.* at 75:21–76:2). Defendant’s requirement that the application be “independent” in addition to merely being “separate” is, Plaintiff maintains, an improper attempt to partition the “application” from the “Player,” especially in light of specification text that, referring to an embodiment, explains that the “application” can be extended “on the Player so that it is efficiently integrated into a comprehensive client/server Application.” (D.I. 64 at 56) (citing ’755 patent at 7:30–33).

Defendant, on the other hand, argues that the code for the “application” and “Player” are “touted as being able to be maintained separately and distinctly,” so they must be not only separate lines of code but also separate files. (D.I. 77 at 80:20–82:18). Defendant also points to references in the prosecution history, including the McCain reference, to highlight that the patentee’s use of the separation of the “application” and “Player” to argue that the invention was novel supports construction of “application” to mean “separate and independent” from the “Player.” (D.I. 64 at 57).

I agree with Plaintiff. Defendant’s argument here—based on the patentee’s assertion of novelty using the separation of the “application” and “Player”—is substantively identical to the argument it made when I issued my prior construction. (D.I. 65-6, Ex. 1E at 7). As I did there, I acknowledge that “separate” should be included in the construction of “application” because the

patentee's assertion that the separation of the "application" and "Player" is novel constitutes prosecution history disclaimer. (*Id.*). In making this novelty argument, the patentee only asserted, however, that the bodies of code for the "application" and "Player" are separate, not that the "application" and "Player" need to be housed independently or distinctly from each other. (*See id.*). Moreover, including "independent" or "distinct" excludes embodiments such as the one highlighted by the specification in which the "Player" is "efficiently integrated" with the "Application." (*Id.*; '755 patent at 7:30–33).

To be sure, the code for the "application" and "Player" are "separate" sets of code. To the extent "independent" or "distinct" are meant to mean something more than "separate," however, the disclaimer does not support that conclusion. Patentee's statements refer only to "partitioning the code" required for the Application and for the Player (i.e. maintaining a separate body of code for each). (D.I. 65-6, Ex. 1E at 7). This "partitioning" is already conveyed by "separate" in the construction. Adding "independent" or "distinct" would imply that there is some other difference between the code for the Application and the code for the Player that goes beyond there merely being "separate" bodies of code. No such additional difference is supported by the patentee's statements during prosecution. Furthermore, since the code for "Application" and "Player" can be integrated on the same server ('755 patent at 7:30–33), adding "independent" or "distinct" to the construction would serve no purpose other than to confuse.

I therefore construe "application" as "device-independent code containing instructions for a device and which is separate from the Player."

## **V. CONCLUSION**

Within five days the parties shall submit a proposed order consistent with this Memorandum Opinion suitable for submission to the jury.

**IN THE UNITED STATES DISTRICT COURT  
FOR THE DISTRICT OF DELAWARE**

---

EXPRESS MOBILE, INC.,

Plaintiff,

v.

GODADDY.COM, LLC,

Defendant.

---

)  
)  
)  
)  
)  
)  
)  
)  
)  
)  
)

Civil Action No.1:19-cv-01937-RGA

**ORDER ON CLAIM CONSTRUCTION**

The following terms and phrases of the patents shall have the following meanings for the purposes of this action:

<b>Claim Term</b>	<b>Patents</b>	<b>Construction</b>
multi-dimensional array(s) / multidimensional array(s)	'397 and '168	a uniquely identifiable indexed set of related elements, wherein each element is addressed by a set of two or more indices, each index corresponding to a dimension of the array
storing information representative of said one or more user selected setting in a database	'397	storing data in a database, which data pertains to one or more attributes of an object available for selection by a user
transformation	'397 and '168	the changing of an object from one state to another based on a timer control, subject to user settings
settings	'397	attributes of an object available for selection
authoring tool / authoring tool configured to	'755, '287 and '044	a system, with a graphical interface, for generating code to display content on a device screen
device-dependent code	'755 and '287	code that is specific to the operating system, programming language, or platform of a device

<b>Claim Term</b>	<b>Patents</b>	<b>Construction</b>
device-independent code	'755 and '287	code that is not specific to the operating system, programming language, or platform of a device
where said application is device-dependent code	'755	where said Application is a device-independent code
for evoking one or more web components	'755, '287 and '044	for calling up one or more web components
at least one run time file / one or more run time files	'397	one or more files, including a run time engine, that are downloaded or created when a browser is pointed to a web page or website
build engine	'397 and '168	not indefinite; "build engine" means "build tool" in the '397 patent; any construction of "build engine" in the '168 patent is deferred
run time (runtime) engine	'168	file that is executed at runtime that reads information from the database and generates commands to display a web page or website
registry	'755 and '287	a database that is used for computing functionality
web component	'755, '287 and '044	software objects that provide functionalities of a web service
web service	'755, '287 and '044	A software system that supports interaction between devices over a network
symbolic name(s)	'755, '287 and '044	No construction necessary
application	'755, '287 and '044	device-independent code containing instructions for a device and which is separate from the Player

DATED: June 7, 2021

BALLARD SPAHR LLP

By: /s/ Brian S.S. Auerbach

Beth Moskow-Schnoll (No. 2900)  
Brittany Giusini (No. 6034)  
Brian S.S. Auerbach (No. 6532)  
**BALLARD SPAHR LLP**  
919 N. Market Street, 11th Floor  
Wilmington, DE 19801-3034  
(302) 252-4465  
moskowb@ballardspahr.com  
giusini@ballardspahr.com  
auerbachb@ballardspahr.com

OF COUNSEL:

**BALLARD SPAHR LLP**

Brian W. LaCorte (*pro hac vice*)  
Jonathon A. Talcott (*pro hac vice*)  
Andrew Michael (*pro hac vice*)  
1 East Washington Street, Suite 2300  
Phoenix, AZ 85004-2555  
(602) 798-5400  
lacorteb@ballardspahr.com  
talcottj@ballardspahr.com  
hensleya@ballardspahr.com

**BALLARD SPAHR LLP**

Shaton C. Menzie  
999 Peachtree Street, NE, Suite 1600  
Atlanta, GA 30309-3915  
(678) 420-9362  
menzies@ballardspahr.com

*Counsel for Defendant/Counterclaimant  
GoDaddy.com, LLC*

DEVLIN LAW FIRM LLC

By: /s/ Timothy Devlin

Timothy Devlin (No. 4241)  
**DEVLIN LAW FIRM LLC**  
1526 Gilpin Avenue  
Wilmington, Delaware 19806  
Tel: (302) 449-9010  
tdevlin@devlinlawfirm.com

*OF COUNSEL:*

James R. Nuttall (*pro hac vice*)  
Michael Dockterman (*pro hac vice*)  
Katherine H. Johnson (*pro hac vice*)  
Robert F. Kappers (*pro hac vice*)  
Tron Fu (*pro hac vice*)

**STEPTOE & JOHNSON LLP**

227 West Monroe, Suite 4700  
Chicago, IL 60606  
(312) 577-1300  
jnuttall@steptoe.com  
mdockterman@steptoe.com  
kjohnson@steptoe.com  
rkappers@steptoe.com  
tfu@steptoe.com

Christopher Suarez (*pro hac vice*)

**STEPTOE & JOHNSON LLP**

1330 Connecticut Avenue, NW  
Washington, DC 20036  
(202) 429-3000  
csuarez@steptoe.com

*Attorneys for Plaintiff Express Mobile, Inc.*

SO ORDERED this 8th day of June, 2021

/s/ Richard G. Andrews

United States District Judge

**IN THE UNITED STATES DISTRICT COURT  
FOR THE DISTRICT OF DELAWARE**

<b>EXPRESS MOBILE, INC.,</b>	)	
	)	
<b>Plaintiff,</b>	)	
	)	
<b>vs.</b>	)	<b>Civil Action No. 1:19-cv-01937-MFK</b>
	)	
<b>GODADDY.COM, LLC,</b>	)	
	)	
<b>Defendant.</b>	)	

**MEMORANDUM OPINION AND ORDER**

MATTHEW F. KENNELLY, District Judge:

Express Mobile, Inc. has sued GoDaddy.com, LLC for infringement of five patents: U.S. Patent No. 6,546,397 (the '397 patent); U.S. Patent No. 7,594,168 (the '168 patent); U.S. Patent No. 9,063,755 (the '755 patent); U.S. Patent No. 9,471,287 (the '287 patent); and U.S. Patent No. 9,928,044 (the '044 patent). The first two patents (collectively, the "Web Design Patents") claim a browser-based tool for building websites. The remaining patents (collectively, the "Web Component Patents") are directed to tools for displaying content on mobile devices. Express Mobile contends that two of GoDaddy's products—the Website Builder (WSB) and Managed WordPress (MWP)—infringe numerous claims of the asserted patents.<sup>1</sup>

GoDaddy has moved for summary judgment on all of Express Mobile's claims and additionally seeks the exclusion of certain testimony by two of Express Mobile's

---

<sup>1</sup> Express Mobile contends that GoDaddy infringes claims 1, 2, 3, 11, and 37 of the '397 patent; claims 1, 2, and 3 of the '168 patent; claims 1, 3, 12, 16, and 22 of the '755 patent; claims 1 and 13 of the '287 patent; and claims 1, 11, 13, 17, and 19 of the '044 patent.



expert witnesses—Dr. Kevin Almeroth and Walter Bratic. Express Mobile has moved for partial summary judgment on certain issues and to exclude certain expert opinions of David Perry. While these motions were being briefed, the parties each filed an additional motion to strike regarding evidence submitted for the first time during summary judgment. In this opinion, the Court rules on all four of these motions and sets forth its construction of certain disputed claim terms.

## **Background**

### **A. The asserted patents**

There are two families of patents at issue in this case. The Web Design Patents claim a browser-based tool for building websites. The two patents share the same specification, which describes a method of website design where a web designer can make selections on various menus, and the system can preview what the page will look like. To do this, attributes of objects on a website (for example, fonts) are stored in an external database. When an end user browses on the website, the user's browser downloads a "run time engine" that reads the database and uses that data to generate the website.

In contrast, the Web Component Patents are directed to displaying content on mobile devices, such as smartphones. The Web Component Patents relate to two concepts that allow web designers to integrate third-party web services into their websites. First, the patent family discloses an authoring tool that generates two sets of code, a device-independent Application and a device-specific Player. Second, the patents disclose an authoring tool that allows for integration of web services into the Application and Player.

**B. The accused products**

Two GoDaddy products are at issue in this case. The WSB is a browser-based website creator, designed to allow users with little to no website design experience to quickly create and publish a website. The MWP, in contrast, targets more advanced users. It comprises a set of custom functionalities that GoDaddy has added to WordPress, an open-source (i.e. free) platform for creating websites. Both the WSB and MWP allow for incorporation of features, such as YouTube videos or embedded Twitter feeds, into a user's website. GoDaddy provides several free themes for use in designing the website, but users can also buy third-party themes for use with the MWP platform.

**C. The instant suit**

Express Mobile filed this lawsuit against GoDaddy on October 11, 2019. The case was assigned to Judge Richard Andrews. On April 8, 2021, the Court held a *Markman* hearing to address several disputed terms of the asserted patents. At the hearing, the Court directed the parties to submit supplemental briefing on claim construction issues pertaining to the terms "virtual machine" and "Player." On June 1, 2021, the Court issued an order construing the other disputed claim terms. *Express Mobile, Inc. v. GoDaddy.com, LLC*, No. 19-1937-RGA, 2021 WL 2209868 (D. Del. Jun. 1, 2021).

On November 17, 2021, both parties filed combined summary judgment and *Daubert* motions. During briefing, each party filed an additional motion to strike evidence that it contended was disclosed for the first time during summary judgment. The case was subsequently reassigned to the undersigned District Judge.

## Discussion

This opinion concerns four motions and two disputed claim terms. The Court starts by construing the disputed claim terms. It then addresses the parties' motions to strike. Lastly, the Court rules on the parties' combined summary judgment/*Daubert* motions.

### A. Claim construction

"[T]he claims of a patent define the invention to which the patentee is entitled the right to exclude." *Innova/Pure Water, Inc. v. Safari Water Filtration Sys., Inc.*, 381 F.3d 1111, 1115 (Fed. Cir. 2004). During claim construction, a court is to construe the words of a claim in accordance with their "ordinary and customary meaning," namely "the meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention." *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312–13 (Fed. Cir. 2005). Sometimes, the meaning of a term is not immediately apparent, and a court will need to look to other sources to determine "what a person of skill in the art would have understood disputed claim language to mean." *Innova*, 381 F.3d at 1116. These sources include "the words of the claims themselves, the remainder of the specification, the prosecution history, and extrinsic evidence concerning relevant scientific principles, the meaning of technical terms, and the state of the art." *Id.*

#### 1. "Virtual Machine"

- a. *Express Mobile's proposed constructions:*
  - i. "software that emulates a physical machine"; or
  - ii. "software that emulates computing functionality"
- b. *GoDaddy's proposed construction:*

- i. "software that emulates a physical processor"
- c. *The Court's construction:*
  - i. "software that emulates computing functionality"

The parties dispute the meaning of the term "virtual machine" in the '397 patent. The '397 patent claims "[a] method to allow users to produce Internet websites on and for computers having a browser and a virtual machine capable of generating displays . . . ." '397 patent at 65:44–46. The invention discloses a "run time file," "where said at least one run time file utilizes information stored in said database to generate virtual machine commands for the display of at least a portion of said one or more web pages." *Id.* at 65:66–66:2.

The construction the Court adopts—"software that emulates computing functionality"—is consistent with the patent's claim language and specification. The patent is directed toward software that emulates computing functionality, specifically, functionality related to the generation of website displays. *See, e.g., id.* at 65:44–46; *id.* at 65:66–66:2. As Express Mobile's experts Bhuvan Urgaonkar and Glenn Weadock explain, the claim language reflects the fact that the virtual machine is used to render and display websites within a browser. Dkt. no. 66-1, ex. 2 ¶ 26; dkt. no. 66-13, ex. 2L ¶¶ 38.

GoDaddy's proposed construction is too narrow. The claims and shared specification indicate that the term "virtual machine" was meant to be interpreted broadly. For example, the specification confirms that virtual machines can support a variety of different programming languages, not just Java, as GoDaddy contends. *See* '397 patent at 11:5–8 (stating that the required functionality "wa[s] implemented entirely

in JAVA (*or any other browser-based full featured programming language*)" (emphasis added). And although the Java Virtual Machine is the only virtual machine listed in the specification, this merely provides an example of a virtual machine and does not limit the claim scope to only such a virtual machine. It is generally inappropriate to use the preferred embodiment outlined in the specification to limit the scope of a patent claim. *Acumed LLC v. Stryker Corp.*, 483 F.3d 800, 805 (Fed. Cir. 2007) (noting that it is improper to "import a feature from a preferred embodiment into the claims").

GoDaddy contends that "virtual machine" is limited to processor functions because the specification describes an interaction between the virtual machine, run time engine, and database where the virtual machine executes processor-like commands from the other two. But nothing in the claims indicates that this is the only function of a virtual machine. Instead, the claim language repeatedly refers to the virtual machine as being capable of generating displays. It would be inappropriate to use another term in the patent, namely "run time engine," to narrow the ordinary meaning of virtual machine.

In addition to being supported by the language of the claims and specification, the Court's adopted construction is consistent with the prosecution history. Dr. Urgaonkar emphasizes that the term was meant to have a broad scope and that the patentability of the invention was not dependent on it possessing a particular kind of virtual machine. Dkt. no. 66-13, ex. 2L ¶ 43. GoDaddy disagrees, arguing that it was the specific processor-like interactions between the virtual machine and the run time engine that distinguished the patented invention from the prior art. The prosecution history suggests otherwise, however. The prior art also disclosed use of a virtual machine—specifically the Java Virtual Machine. As such, the addition of the term virtual

machine could not have distinguished the patents from the prior art.

The Court's construction is also supported by extrinsic evidence of what a person of ordinary skill in the field would have understood the term to mean. Express Mobile provides a variety of evidence to establish that, as of the priority date, the term virtual machine was understood to refer to several different kinds of software. See *dk. no. 120* at 8–13. Express Mobile cites dictionary definitions, lecture notes, and textbook entries—all of which support a broad construction of the term that encompasses more than processor-like virtual machines. *Id.*

GoDaddy makes several objections to Express Mobile's evidence. Some of these have merit. For example, the Court agrees that the lecture notes and any evidence drafted after the priority date are irrelevant to the claim construction analysis. It does not agree, however, that Express Mobile's evidence demonstrates that the meaning of "virtual machine" was "all over the map" and had no well-known meaning in 1999. *Id.* at 28. Although the evidence does show that the definition was broad, it does not show that the term was unclear. Considering all of the relevant extrinsic evidence, the Court concludes that, in 1999, virtual machines were understood to come in many different types but were all understood to emulate computing functionality.

GoDaddy makes three final arguments against Express Mobile's (and the Court's) construction. First, GoDaddy argues that the construction assumes that a virtual machine is the same as the separately claimed browser. Not so. As Express Mobile explains, the patent discloses a virtual machine that works *with* the browser to generate a display. Specifically, the virtual machine enhances browsers that do not support certain kinds of programming languages. So, although the two are related, the

construction does not assume they are the same.

Next, GoDaddy argues that Express Mobile's construction inappropriately claims innovations that were developed after the patents were issued. This argument, however, misinterprets the adopted construction. Express Mobile's evidence shows that, at the priority date, there were many different kinds of virtual machines. Any of these virtual machines (and others developed after the priority date) could be used in the patented invention. But compatibility with newer-developed virtual machines does not amount to a patent of those virtual machines. As Express Mobile explains, if anything it "underscore[s] that the virtual machine in the '397 patent is nothing more than an off-the-shelf component that is used to properly display the claimed functionality within the browser." Dkt. no. 120 at 48.

GoDaddy's final argument is also unpersuasive. It contends that the Court's construction renders "virtual machine" a so-called nonce word, and as such, the Court must limit the term to a specific kind of virtual machine, namely, the Java Virtual Machine. Under patent law, claims involving nonce words are means-plus-function claims that must meet the additional requirements in 35 U.S.C. § 112. See *Williamson v. Citrix Online, LLC*, 792 F.3d 1339, 1348 (Fed. Cir. 2015). When considering a means-plus-function claim, a court must identify both the claimed function and the corresponding structure in the written description for performing that function. *Id.* GoDaddy contends that there is no claimed structure in the construction "software that emulates computing functionality" and thus the claim must be confined to the Java Virtual Machine. The Court disagrees. Its adopted construction of "virtual machine" does not render it a nonce word. First, as previously explained, the term had a well-

established meaning as of the priority date and would have been understood as such by a person of ordinary skill in the field. Additionally, the adopted construction provides sufficient structure through the use of the word "software" and also explains what the software does, namely "emulates computing functionality." See, e.g., *WhitServe LLC v. GoDaddy.com, Inc.*, 65 F. Supp. 3d 317, 321 (D. Conn. 2014) (noting that "[software] is a noun with a specific structural meaning, defining the set of coded instructions and programs governing the operation of computer hardware").

## **2. "Player"**

### *a. Express Mobile's proposed construction:*

- i. "software code that facilitates the execution of an application on a device"

### *b. GoDaddy's proposed constructions:*

- i. "device-specific code which contains instructions of a device and which is separate and independent from the Application"
- ii. "device-specific code which contains instructions for a device and which is separate and distinct from the Application"

### *c. The Court's construction:*

- i. "device-specific code which contains instructions of a device and which is separate from the Application"

The parties dispute the meaning of the term "Player" as it appears in the '044 patent. There are two points of disagreement. First, the parties dispute whether "Player" refers to device-specific code in the '044 patent. They agree that the term when used in the '755 and '287 patents refers to device-specific code, but Express



Mobile contends that the term as used in the '044 patent is different and does not require device-specific code. Second, the parties' constructions differ in that GoDaddy's constructions state that the Player and Application are either "separate and independent" or "separate and distinct," whereas Express Mobile's construction does not require separation of the Player and Application.

On the first issue, the Court concludes that "Player" in the '044 patent refers to device-specific code. The issue of whether "Player" necessarily refers to device-specific code previously came up in a related case, *Shopify Inc. v. Express Mobile, Inc.*, CA No. 19-439-RGA, 2020 WL 3432531 (D. Del. June 23, 2020). In adopting the same construction as it does here, the Court (Judge Andrews) concluded that "a person of ordinary skill in the art, looking at this intrinsic evidence as a whole, would understand the Player to be 'device-specific.'" *Id.* at \*6. The Court noted that the shared specification of the Web Component Patents repeatedly recites a Player with device-specific instructions. *Id.* at \*5 (citing the '755 patent at 5:8-24; 23:43-46; 33:12-15; 33:26-28).

Additionally, the Court in *Shopify* found that "the applicant made 'repeated and consistent remarks' indicating that the Player was device-specific" during patent prosecution. *Id.* at \*6 (quoting *Personalized Media Commc'ns, LLC v. Apple Inc.*, 952 F.3d 1336, 1340 (Fed. Cir. 2020)). For example, the patentee stated:

One embodiment is an authoring tool that generates files which together provide content over a network. The files include a Player (sometimes referred to herein as a "first code") specific to each device (that is, the code is "device-dependent") and an Application (sometimes referred to herein as a "second code") that is device-independent.

*Id.* (citation omitted). The patentee's statement suggests that he believed the

device-specific nature of the Player to be a central component of the invention's patentability. This further supports the Court's adopted construction. See *Personalized Media*, 952 F.3d at 1340 ("[A] applicant's repeated and consistent remarks during prosecution can define a claim term by demonstrating how the inventor understood the invention.").

Express Mobile argues that the '044 patent specification supports its construction, citing two passages. The first passage, in its entirety, reads: "In one embodiment, one of the codes is a Player, which is a thin client architecture that operates in a language that manages resources efficiently, is extensible, supports a robust application model, *and has no device specific dependencies*." '044 patent at 1:55–67 (emphasis added). Express Mobile contends that the phrase "has no device specific dependencies" refers to the Player, lending support to its contention that a Player need not be device specific. GoDaddy disagrees. It contends that "has no device specific dependencies" refers to "thin client architecture" and does not suggest that a Player can be device-independent. Both interpretations are plausible grammatically, and neither party provides the Court with a sufficiently persuasive reason to choose one interpretation over the other. As such, the Court gives this language little weight and looks instead to the other language in the claims and specification.

The second passage that Express Mobile cites is "The Player may include code that is device-specific . . . ." '044 Patent at 6:14–16. The Court agrees with Express Mobile that the patent's use of the term "may" instead of "must" lends some support to Express Mobile's construction, but it is important not to overstate its significance. The drafter's intention behind the word "may" is less than clear. It could be the result of

sloppy drafting; it could have no intended meaning at all. Either way, the Court finds that this "single reference . . . is not enough to override the repeated references in the prosecution history and the specification to the 'device-specific' or 'device-dependent' Player." *Shopify*, 2020 WL 3432531, at \*6. On balance, the claims, specification, and prosecution history of the patents indicate that "Player" must be device specific.

Express Mobile also makes several arguments contending that the claims, specifications, and prosecution history of the '755 and '287 patents should not be used to interpret the '044 patent. It identifies three distinctions between the first two patents and the third that it contends justify a different construction for the disputed term. First, Express Mobile points to the fact that the claims of the '044 patent do not recite "Player" as device-dependent code, whereas the claims of the '755 and '287 patents do. The prosecution history, however, shows that this omission was not meant to change the meaning of the patents. When the claims were amended to omit the "device-dependent" language, the patentee claimed that this was "[n]o new matter." Dkt. no. 120, ex. 10D at XMO–GD00003351. Additionally, the '044 patent claims do not state that the Application is "device-independent" code, yet both parties agree that the Application is device-independent code. See '044 patent at 38:20–28. The Court thus concludes that the omission of "device-dependent code" is not meant to suggest that the Player can be device-independent.

Second, Express Mobile argues that the patents should be interpreted differently because the '755 and '287 patents use the term "registry," whereas the '044 patent uses the term "database." This is not a significant difference. The function of a database and registry are the same, the information in each is the same, and Express Mobile

concedes that a registry can be a database.

Lastly, Express Mobile contends that the difference in capitalization use between the '755 and '287 patents and the '044 patent suggests that the terms should not be construed the same. Not so. As GoDaddy points out, the patentee used capital and lower-case letters interchangeably in the shared specification when referring to the Application and Player. There is no support for the proposition that the use of capital versus lower-case letters was intended to be significant. For these reasons, the Court concludes that Player must refer to device-specific code.<sup>2</sup>

On the second issue—whether to include the "separate and independent" or "separate and distinct" language—the Court finds that the construction should include the term "separate" but not "independent" or "distinct." As previously explained in *Shopify*, the patentee emphasized the separate nature of the Application and Player codes during patent prosecution. *Shopify*, 2020 WL 3432531, at \*4. For example, the patentee stated: "[T]he claimed invention, in contrast, operates by partitioning the code required for functionality into device-independent code and device-dependent code." *Id.* (citation omitted). This partitioned code was described as an "advantage for maintaining websites." *Id.* (citation omitted). Because the separate nature of the Player and Application codes was critical to the novelty of the patent, it must be reflected in the construction.

Express Mobile argues that incorporation of "separate" into the construction is

---

<sup>2</sup> GoDaddy further contends that Express Mobile is estopped from arguing that the term "Player" in the '044 patent is not device specific given the patentee's representations during the '755 patent prosecution. The Court notes that the argument is plausible but declines to address it, as it is not necessary for the resolution of the issue.

inconsistent with the specification, which states that the Application can be integrated with the Player. The Court disagrees. Even if the Application is integrated with the Player, it still constitutes separate code from the player. The Court thus agrees with GoDaddy that the Application and Player codes must be "separate."

The Court declines to adopt GoDaddy's "independent" or "distinct" language, however. As Express Mobile points out, the term "independent" is misleading because it incorrectly suggests that the Player and Application do not work together. See '755 patent at 8:27–35, 13:46–49; *id.* at 5:32–41. Additionally, GoDaddy's proposed constructions could be potentially confusing to a jury. "Independent" and "distinct" seem to be synonyms of "separate," and GoDaddy does not explain what function adding these terms would serve that the word "separate" does not already provide. The Court thus declines to adopt this part of the proposed construction.

#### **B. Motions to strike**

Both parties have filed motions to strike evidence that they contend was disclosed for the first time during summary judgment briefing. Express Mobile's motion concerns declarations by GoDaddy engineers Franklin Jarrett and Aaron Silvas that were filed as exhibits to GoDaddy's motion for summary judgment. GoDaddy's motion concerns a supplemental report by Express Mobile's expert Kevin Almeroth that was filed as an exhibit to Express Mobile's response to GoDaddy's summary judgment motion.

Federal Rule of Civil Procedure 26 sets out the default guidelines for parties' pretrial disclosures. Fed. R. Civ. P. 26. It requires certain pretrial disclosures to be made at least 30 days before trial, "[u]nless the court orders otherwise." Fed. R. Civ. P.

26(a)(3). It also sets out the rules concerning expert discovery. Under Rule 26(a)(2)(B), the disclosure of an expert witness must be accompanied by a written report containing a statement of all opinions the witness will express at trial, as well as all of the facts and data underlying the witness's opinions. A party who fails to disclose a witness as required by Rule 26 can be precluded from relying on that evidence on a motion or at trial unless the failure to disclose was "substantially justified or harmless." Fed. R. Civ. P. 37(c)(1). Rule 37(c)(1) also permits a court to decline to exclude the evidence and instead impose "other appropriate sanctions." *Id.*

In the Third Circuit, courts look to the so-called *Pennypack* factors to determine whether to exclude evidence as a sanction. *Meyers v. Pennypack Woods Home Ownership Ass'n*, 559 F.2d 894, 904 (3d Cir. 1977). These factors are: (1) prejudice or surprise to the other party; (2) the ability of the injured party to cure the prejudice; (3) the likelihood of disruption to the trial schedule; (4) any bad faith or willfulness involved in not complying with the disclosure rules; (5) the violating party's explanation for noncompliance; and (6) the importance of the evidence to the party offering it. *Id.* Although the Third Circuit has stated that the last factor is the most important, it has advised that courts must look to the "overall balance" of the factors. *ZF Meritor, LCC v. Eaton Corp.*, 696 F.3d 254, 299 (3d Cir. 2012); *see also, e.g., Wyeth Holdings Corp. v. Sandoz, Inc.*, No. 09-955-RGA-CJB, 2012 WL 1669555, at \*4 (D. Del. May 10, 2012).

#### **1. Express Mobile's motion to strike**

In support of its summary judgment motion, GoDaddy filed declarations from Silvas and Jarrett, two of its engineers who worked on the WSP and MWP, respectively. Express Mobile argues that this disclosure ran afoul of Rule 26, for two reasons. First, it

argues that GoDaddy failed to set forth the facts alleged in the declarations during fact or expert discovery, thus failing to comply with the deadlines in Rule 26(a), Rule 26(e), and the Court's scheduling order. Second, Express Mobile contends that the declarations contain expert testimony that was not properly or timely disclosed.

On the first point, the Court agrees with Express Mobile. The parties completed fact discovery on June 16, 2021, and expert discovery on November 3, 2021. During discovery, Express Mobile sought GoDaddy's bases for noninfringement. One of its interrogatories stated:

For each claim of the patents in suit that you assert you do not infringe, describe in detail the factual and legal bases for your assertion, including providing a claim chart for each Accused Instrumentality showing on a limitation-by-limitation basis any limitations that you believe are not satisfied, and identifying all evidence upon which you rely and the persons with relevant knowledge.

Pl's Int. No. 8. In response to this request, GoDaddy stated that it would respond through its expert disclosures and did not otherwise disclose any bases for noninfringement. None of the expert disclosures, however, disclosed the facts included in the disputed declarations. And it was not until November 17, 2021, long after the close of fact and expert discovery, that GoDaddy disclosed the disputed declarations. This violated the Court's scheduling order and GoDaddy's continuing disclosure obligations set out in Rule 26(e).

GoDaddy's attempts to justify this delay are unpersuasive. First, GoDaddy argues that the declarations are timely because they rely on evidence previously disclosed to Express Mobile. Although it is true that Silvas's and Jarrett's new testimony relies on evidence, such as source code, that was previously disclosed, the declarations still provide new facts and arguments derived from that evidence that

cannot fairly be said to be restatements of that previously disclosed evidence.

GoDaddy further argues that, even if the declarations were untimely, this delayed disclosure was substantially justified because of Express Mobile's actions. Specifically, GoDaddy contends that Express Mobile shifted infringement positions in its expert reply report and second amended infringement contentions. It contends that this change in positions was unforeseeable, so it could not have previously disclosed facts necessary to address the new position.

The Court agrees that Express Mobile's position in these later filings did differ from its position in its earlier expert reports. It appears that this shift in position was in response to the ruling in *Shopify Inc. v. Express Mobile, Inc.*, No. 19-439-RGA, 2021 WL 4288113 (D. Del. Sept. 21, 2021). Although Express Mobile's expert discussed the instantiation theory in his opening report, his explanation was relatively cursory, and it was not until the reply expert report that any actual evidence was cited in support of this theory.

This explanation does not get GoDaddy off the hook, however. The need to address Express Mobile's new instantiation theory evidence is only a partial explanation for why GoDaddy proceeded with the declarations as it did. It explains why GoDaddy offered the new declarations but not why it waited until summary judgment to do so. Express Mobile's second amended infringement contentions and expert reply report were disclosed on June 16, 2021 and October 1, 2021, respectively, well over one month before the end of expert discovery. GoDaddy thus had plenty of time to disclose these declarations or seek an extension of the discovery deadline. It did neither. For this reason, the Court finds that GoDaddy's delayed disclosure of the Silvas and Jarrett



declarations was not substantially justified, and a sanction is warranted.<sup>3</sup>

Noncompliance with discovery deadlines does not automatically result in the striking of the untimely evidence. To determine whether an untimely declaration should be stricken, courts in the Third Circuit apply the above-listed *Pennypack* factors.

**a. Prejudice**

GoDaddy's late disclosure of the Silvas and Jarret declarations caused prejudice to Express Mobile. Because the declarations were filed after the fact and expert discovery deadlines, Express Mobile was not able to depose the witnesses or conduct additional discovery to address the new testimony. The Court notes that Express Mobile asked to depose Silvas and Jarrett after the disclosure of their declarations, but GoDaddy refused. GoDaddy contends that Express Mobile was not prejudiced because it previously deposed the declarants. This argument is unpersuasive, however, because Express Mobile did not have the benefit of having the declarations prior to or during these depositions. At that time, Express Mobile did not know that the declarants would be used as support for the defendant's non-infringement arguments and thus could not effectively question them on the issue.

**b. Opportunity to cure and disruption to the trial schedule**

The trial is currently set for February 27, 2023, which gives the parties plenty of time to conduct additional discovery on the issue.

**c. Explanation for failure to disclose**

GoDaddy's explanation for its delayed disclosure is that Express Mobile changed

---

<sup>3</sup> Because the Court agrees that the declarations were not timely disclosed, it is unnecessary for it to analyze Express Mobile's second argument regarding whether the declarations contain expert testimony.

its infringement position, creating the need for GoDaddy to respond with the disputed declarations. As previously explained, however, Express Mobile disclosed its new instantiation theory more than a month before the end of the discovery deadline. GoDaddy fails to explain why it waited until summary judgment to offer the evidence in question.

**d. Willfulness or bad faith**

Aside from the violation itself, there is no evidence of willfulness or bad faith on the part of GoDaddy.

**e. Importance of evidence**

This last factor is the most important in determining whether to strike untimely evidence. *ZF Meritor*, 696 F.3d at 299. The Court finds that the disputed declarations are critical to GoDaddy's noninfringement contentions. Because Express Mobile did not fully flesh out its instantiation theory until its later disclosures, the Silvas and Jarrett declarations are the only evidence that directly addresses these new arguments.

\* \* \*

Considering the factors together, the Court declines to strike the declarations. Although GoDaddy's delayed disclosure resulted in prejudice to Express Mobile, this prejudice can be cured by allowing Express Mobile to depose the declarants on the points in their declarations. There is no evidence of bad faith, and the weightiest factor, the importance of the evidence, points in favor of keeping the declarations in. Instead of striking the evidence, the Court instead orders GoDaddy to make Silvas and Jarrett available for deposition within the next thirty days, with the depositions limited to 90 minutes for each. GoDaddy will also be required to pay for three hours of Express

Mobile's attorney's fees, as well as the court reporter's appearance fees for the two depositions.

**2. GoDaddy's motion to strike**

In response to GoDaddy's summary judgment motion and accompanying declarations, Express Mobile filed a supplemental report by its expert witness Dr. Kevin Almeroth. GoDaddy seeks to strike this report, contending that it discloses yet another new theory of infringement. The Court agrees with GoDaddy that the supplemental report presents a new version of the instantiation theory that was not previously disclosed. After considering the *Pennypack* factors, however, it concludes that exclusion is not warranted.

**a. Prejudice**

GoDaddy is prejudiced by the late disclosure of a new theory of infringement because it was not able to address it with its own expert report and because it was not able to conduct additional discovery to respond to the point.

**b. Opportunity to cure and disruption to the trial schedule**

There is an opportunity to cure the prejudice before trial, which is scheduled for February 2023.

**c. Explanation for failure to disclose**

The Court finds Express Mobile's explanation for its late disclosure persuasive. Because GoDaddy filed the Silvas and Jarrett declarations during summary judgment, Express Mobile needed to submit the supplemental report to cure the prejudice caused by this late disclosure.

**d. Willfulness or bad faith**

The Court finds no evidence of willfulness or bad faith. It rejects GoDaddy's argument that the fact that Express Mobile is represented by competent counsel suggests willfulness. Just because a party is represented by competent counsel does not mean that a discovery violation is willful.

**e. Importance of evidence**

The parties agree that the evidence is important, but they disagree over the implication of this fact. Contrary to its statements in its other filings, GoDaddy now argues that the importance of a piece of evidence weighs in favor of *exclusion*. Compare *dkt. no. 213* at 18 (stating that "[i]mportance here weighs against exclusion"), with *dkt. no. 251* at 9 (arguing that Third Circuit precedent supports "excluding 'critical' evidence where disclosure deadlines have been flouted"). GoDaddy's new argument is unpersuasive. As previously stated, the law in the Third Circuit is that the importance of the evidence weighs *against* exclusion. See, e.g., *ZF Meritor*, 696 F.3d at 299. The importance of the report thus indicates that the Court should impose a remedy that addresses the prejudice without totally excluding the evidence.

\* \* \*

Overall, the Court concludes that the factors weigh against striking the supplemental report. There is prejudice to GoDaddy, but this can be cured given the long amount of time before trial. Express Mobile's explanation for its failure to disclose is reasonable, and there is no evidence of bad faith. Lastly, the weightiest factor, the importance of the evidence, points in favor of keeping the report in. Instead of striking the report, the Court allows GoDaddy to depose Dr. Almeroth for no more than 90

minutes on his new testimony. The parties are responsible for their own costs and fees, with Express Mobile covering Dr. Almeroth's expert fee relating to his appearance at the deposition (but not for preparation).

**C. GoDaddy's combined summary judgment/*Daubert* motion**

**1. Motion for summary judgment**

Summary judgment is appropriate if "the movant shows that there is no genuine dispute as to any material fact." Fed. R. Civ. P. 56(a). There is a genuine issue of material fact if "the evidence is such that a reasonable jury could return a verdict for the nonmoving party." *Jutrowski v. Township of Riverdale*, 904 F.3d 280, 288 (3d Cir. 2018) (citing *Anderson v. Liberty Lobby, Inc.*, 477 U.S. 242, 248 (1986)). In evaluating the summary judgment motion, courts must view all facts "in the light most favorable to the non-moving party, with all reasonable inferences drawn in that party's favor." *Id.* (citation omitted).

**a. Infringement of the '397 and '168 patents**

Claims 1, 2, 3, 11, and 37 of the '397 patent each require a "run time file," and claims 1, 2, and 3 of the '168 patent each require a "run time engine." According to this Court's claim constructions, a "run time engine" is a "run time file" that must meet two requirements: (1) it must be "downloaded or created when a browser is pointed to a web page or website" and (2) it must "read[] information from the database." Dkt. no. 129 at 2. GoDaddy contends that it is entitled to summary judgment on the '397 and '168 patent claims because the accused products do not meet the "run time engine" limitations of these claims.

Express Mobile contends that four categories of files in the accused products

constitute run time engines: (1) preview render files; (2) DPS render files; (3) JavaScript sent to customers' browsers; and (4) PHP templates. Preview render files are files on the WSB server that are triggered when a GoDaddy customer presses the button to "preview" the website he is creating. DPS render files are files on GoDaddy's Dynamic Page Server (DPS) that handle dynamic content—for example, from Twitter or Yelp—that is embedded in a website. Category 3 includes the JavaScript in files sent to a visitor's browser to build the web display in the browser (a visitor is a person browsing the GoDaddy customer's published website). Lastly, PHP templates are files used to create themes for websites designed using the MWP.

GoDaddy disputes that these four categories of files meet either requirement of a run time engine.

**i. Downloaded or created**

Express Mobile argues that the files in Categories 1, 2, and 4 meet the "downloaded or created" requirement because they are created at runtime. Specifically, it contends that, at runtime, an "instance" of the files is created in the RAM memory of GoDaddy's server from the static code files stored on GoDaddy's server disk and that this "instantiation" meets the "created" requirement. In support of its instantiation theory, Express Mobile cites testimony from its experts, documents, and source code.

GoDaddy argues that Express Mobile's cited evidence is insufficient to support its instantiation theory. It contends that Express Mobile fails to cite evidence specific to its source code or products and instead cites evidence that talks only generally about instantiation. The Court agrees. Almeroth fails to identify any actual line of code that represents the "instantiated" version of the accused run time engine files. The other

evidence that Express Mobile cites is either outdated, not specific to GoDaddy or the accused products, not specific to run time engine files, or a combination of the three.

Although the files in Categories 1, 2, and 4 do not meet the "downloaded or created" requirement, the Court finds that those in Category 3 do. The parties agree that the files in Category 3 are downloaded from the Content Delivery Service (CDN) to the customer's browser. GoDaddy's only argument is that this does not meet the requirement because the code is not downloaded directly from the DPS database. But nowhere does GoDaddy point to language in the claim requiring the run time engine to be downloaded directly from its servers. In fact, the language of the claim states only that the run time engine must be downloaded to a visitor's browser when the browser requests a website. On this question, Express Mobile has adduced sufficient evidence to create a genuine factual dispute.

GoDaddy also argues that the downloaded JavaScript code cannot constitute a run time engine because it also contains the "virtual machine commands." GoDaddy contends that Express Mobile's position here creates the same "tension" that the Court recognized in Express Mobile's position in *Shopify*. See *Shopify*, 2021 WL 4288113, at \*9. Express Mobile disputes whether the JavaScript run time engine is the same file that contains the "virtual machine commands." It argues that the JavaScript run time engines, such as script.js, "are distinct JavaScript files that are downloaded to a visitor's browser and when executed in the browser's JavaScript engine generate virtual machine commands in the form of additional HTML, CSS, or JavaScript code that update the webpage display." Dkt. no. 191 at 7. This is supported by Dr. Almeroth's opening report, Almeroth Opening Report ¶¶ 345–46, and is sufficient to create a

genuine factual dispute on the issue.

**ii. Reads information from the database**

The second requirement of a run time engine is that it "reads information from the database." Because the Court concludes that the files in Categories 1, 2, and 4 do not meet the first requirement of a run time engine, it need not determine if they meet the second requirement. With respect to Category 3, Express Mobile argues that the files "read information from a database by fetching information from the database over the network." Dkt. no. 191 at 10. It cites Dr. Almeroth's testimony identifying the function "fetch()" in GoDaddy's source code, which he contends causes information in the database to be downloaded to the browser at runtime. Almeroth Opening Report ¶¶ 344–46; Almeroth Reply Report ¶¶ 200, 220–21.

GoDaddy argues that the fetch() function does not "read information from the database" because the files rely on several intermediary files to access the database rather than reading the database directly. The Court agrees. Express Mobile's position here is similar to its position in *Shopify*. There, Express Mobile conceded that the alleged run time engine did not read the database directly but rather "triggered the drop file to perform its data fetching function." See *Shopify*, 2021 WL 4288113, at \*10. "Because Express Mobile agrees that the drop file is doing the actual reading and is separate from the claimed run time engine," the Court concluded, "Express Mobile has not shown that the accused Liquid template file is reading from the database." *Id.*

The same logic applies here. Express Mobile provides no evidence that the files in Category 3 actually read the database. Instead, the evidence shows that these files interact only with the CDN and rely on other files to call upon the database. This is not



enough. Any other interpretation would over-expand the requirement to encompass all of the numerous files in the "pipeline" of files. Dkt. no. 206 at 5. It would include the accused file, the file that the accused file invokes, the file that that file invokes, and so on.

Lastly, Express Mobile attempts to argue that even if the accused run time engines do not literally read information from the database, it does the equivalent of reading information from the database. Not so. Under Federal Circuit precedent, there can be no infringement under the doctrine of equivalents if the "theory of equivalence would vitiate a claim limitation." *Tronzo v. Biomet, Inc.*, 156 F.3d 1154, 1160 (Fed. Cir. 1998). Here, Express Mobile's doctrine of equivalence argument vitiates the "reads from" requirement from the claim. Contrary to Express Mobile's contention, accessing a database by using an API or library is not the equivalent of reading a database. As GoDaddy's evidence demonstrates, the accused run time engines "do not read from or contain any references to the source of the data at all, whether through a database-access library or otherwise." Dkt. no. 163 at 16 (citing Jarrett Decl. ¶¶ 10–29; Silvas Decl. ¶¶ 14–17, 21–30, 34–40). And there is no support for the proposition that a file that is filled in by another program with information that comes from a database works in substantially the same way as a file that reads from a database. Express Mobile's theory seems to mimic its previously rejected claim construction argument that an accused run time engine need only "facilitate the retrieval of information" from the database instead of "read from" the database. *Shopify*, 2020 WL 3432531, at \*3. Express Mobile cannot appropriately use the doctrine of equivalence to attempt to relitigate its unsuccessful claim construction positions. See *Augme Techs., Inc. v.*

*Yahoo! Inc.*, 755 F.3d 1326, 1335 (Fed. Cir. 2014) ("The concept of equivalency cannot embrace a structure that is specifically excluded from the scope of the claims.") (citation omitted). For these reasons, the Court grants summary judgment in favor of GoDaddy with respect to infringement of the '397 and '168 patent claims.

**b. Infringement of the '755, '287, and '044 patents**

Each asserted independent claim in the '755, '287, and '044 patents requires the accused products to "produce a Player." As stated above, the Court has construed the term "Player" as "device-specific code which contains instructions of a device and which is separate from the Application." "Device-specific code," or "device-dependent code," has been previously construed in this case to mean "code that is specific to the operating system, programming language, or platform of a device." Dkt. no. 129 at 1. Additionally, the accused Player must "receive[] the output symbolic name [generated by the web service] and corresponding one or more output values and provide[] instructions for a display of the device to present an output value in the defined UI [user interface] object." '755 patent, claim 1; '287 patent, claim 1; '044 patent, claim 1.

Express Mobile contends that the accused products contain a "Javascript Player" that "spans multiple files . . . that work together to meet the claims as construed by the Court." Dkt. no. 191 at 15. GoDaddy makes two arguments in response: (1) the accused Javascript Player does not "receive[] the output symbolic name . . . to present an output value in the defined UI object" and (2) the output values are not displayed "via 'device-dependent' code." Dkt. no. 163 at 22–23. Neither argument is persuasive.

On the first point, GoDaddy contends that a Player must receive an output symbolic name and display that same output value to the end user. It concedes that the

accused Player receives and displays output values, but it argues that the accused Player does not display the *correct* value because it is not the same value as the value received. But nothing in the language of the claim requires that the received and displayed output values be the same. See, e.g., '755 patent at 38:41–57. The claim states that the Player presents "*an* output value," not "*the* output value" or "*said* output value." The use of the indefinite article rather than these other alternatives indicates that the Player can display any output value. It is thus sufficient, as Express Mobile contends, that the output symbolic name results in a corresponding output value, even if the value is different.

On the second point, GoDaddy argues that Express Mobile fails to identify any specific part of the JavaScript Player code that constitutes device-dependent code that results in output values being displayed to the end user. It does not dispute that the JavaScript Player code generally contains device-dependent code, but it argues that such code spans across several files and that Express Mobile does not show that the portions of the code that are device dependent are responsible for displaying output values to the end user.

Again, this issue was addressed in *Shopify*, and the Court adopts the same reasoning as in that case. See *Shopify*, 2021 WL 4288113, at \*17. Express Mobile has provided sufficient evidence to create a genuine factual dispute on the issue. It points to several files that perform output display functionality that work with other files that contain conditional branching. See dkt. no. 191 (discussing the `www-embed-player.js` and `jquery.min.js` files). For these reasons, summary judgment regarding infringement on these claims is inappropriate.

**c. Willfulness**

Although patent infringement is a strict liability offense, enhanced damages are available if the defendant's infringement was "willful." See *XY, LLC v. Trans Ova Genetics, L.C.*, 890 F.3d 1282, 1286 (Fed. Cir. 2018). A plaintiff shows willful infringement if it meets the following elements: "(1) [it] knew of the patent-in-suit; (2) after acquiring that knowledge, it infringed the patent; and (3) in doing so, it knew, or should have known, that its conduct amounted to infringement of the patent." *Välinge Innovation AB v. Halstead New Eng. Corp.*, No. 16-1082-LPS-CJB, 2018 WL 2411218, at \*13 (D. Del. May 29, 2018). GoDaddy contends that Express Mobile cannot meet these requirements.

In support of its contention that GoDaddy willfully infringed, Express Mobile provides two categories of evidence: (1) testimony that someone representing GoDaddy admitted that GoDaddy infringed the asserted patents and (2) a string of e-mails sent in 2013 and a 2018 letter from Express Mobile to GoDaddy regarding the '397 and '168 patents. Regarding the first category, GoDaddy contends that the testimony alleging that a GoDaddy representative admitted infringement is inadmissible as hearsay. The Court agrees. Contrary to Express Mobile's contentions, the testimony does not meet the requirements of Federal Rule of Evidence 801(d)(2)(D) for agent-admissions. The deponent was unable to provide any information about the declarant. He merely states that the admission was made by someone who represented GoDaddy for a possible purchase acquisition and that the declarant might have been the CTO. Without more, this is not sufficient information to establish that the testimony falls within the Rule. See *Carden v. Westinghouse Elec. Corp.*, 850 F.2d 996, 1001–02 (3d Cir.

1988).

The cited e-mails and letter, however, are admissible and sufficient to create a genuine factual dispute with respect to the '397 and '168 patents.<sup>4</sup> The e-mail, sent to GoDaddy in 2013, notified GoDaddy of the asserted patents and contained claim charts of WordPress. Although the e-mail did not explicitly accuse GoDaddy of infringement, such an inference could be inferred from the exchange. Additionally, in 2018, Express Mobile sent GoDaddy a letter in which it stated that it "believed that GoDaddy, Inc. has infringed and is infringing one or more of the Express Mobile patents through its business of building web sites and web pages for customers." Dkt. no. 162-17 at ECF p. 3 of 3. It again informed GoDaddy of the '397 and '168 patents and referenced the 2013 e-mails. *Id.* Considered together, these communications are sufficient to create a genuine factual dispute concerning willful infringement.

**d. Direct infringement**

Direct infringement occurs where all steps of a claimed method or all elements of a claimed system are performed by or attributable to a single entity. *Akamai Techs., Inc. v. Limelight Networks, Inc.*, 797 F.3d 1020, 1022–23 (Fed. Cir. 2015); *Centillion Data Sys., LLC v. Qwest Communs. Int'l*, 631 F.3d 1279, 1284 (Fed. Cir. 2011). For method claims, a single entity can be held liable for the infringement even where more than one actor is involved in practicing the steps if that "entity directs or controls others' performance." *Akamai*, 797 F.3d at 1022. For system claims, if the asserted claims encompass user conduct, then there is no direct infringement of the claims. See

---

<sup>4</sup> Express Mobile does not contend—and the evidence does not support—that this evidence informed GoDaddy of the possibility that it infringed the '755, '287, or '044 patents.

*Centillion*, 631 F.3d at 1284.

GoDaddy contends that the asserted claims encompass user conduct. If that is the case, it contends, then under *Centillion* the Court should grant summary judgment in its favor on the direct infringement system claims. Additionally, if the asserted claims encompass user conduct, GoDaddy reasons, the Court should grant summary judgment in its favor on the direct infringement method claims because Express Mobile cannot meet its burden under *Akamai* to show that GoDaddy "directs or controls" the conduct of the customers.

The Court disagrees that the claims encompass user conduct. The claims are structured to focus on the actions of the program provider, not the customer. See *Uniloc USA, Inc. v. Microsoft Corp.*, 632 F.3d 1292, 1309 (Fed. Cir. 2011) ("A patentee can usually structure a claim to capture infringement by a single party, by focusing on one entity."). Although the claims contain descriptive language involving the end user, user conduct is not necessary for infringement. Rather, the user language merely describes the components of the patented product. This case is thus unlike *Centillion*, in which the plaintiff conceded that part of the claim included a "'front-end' system maintained by an end user." *Centillion*, 631 F.3d at 1281.

Because the claims do not encompass user conduct, *Centillion* does not control, and the divided infringement test outlined in *Akami* does not apply. These two contentions underlie the entirety of GoDaddy's direct infringement summary judgment argument. Accordingly, GoDaddy is not entitled to summary judgment on Express Mobile's direct infringement claims.

**e. Indirect infringement**

"To prevail under a theory of indirect infringement, [the plaintiff] must first prove that the defendants' actions led to direct infringement of the [asserted] Patent."

*Dynacore Holdings Corp. v. U.S. Philips Corp.*, 363 F.3d 1263, 1272 (Fed. Cir. 2004).

35 U.S.C. 271 outlines two methods by which a defendant can indirectly infringe on a patent: (1) induced infringement and (2) contributory infringement. *Id.* at §271(b)–(c).

**i. Induced infringement**

Induced infringement under 35 U.S.C. § 271(b) requires that the plaintiff show "(1) direct infringement of the asserted patents; (2) [] the defendant knew of the patent; (3) [] the defendant knew or should have known that the induced acts constitute patent infringement, and (4) [] the defendant possessed specific intent to encourage another's infringement." *Sanofi, LLC v. Watson Labs. Inc.*, 875 F.3d 636, 643–44 (Fed. Cir. 2017). Specific intent means an "intent to encourage infringement." *Takeda Pharms. v. West-Ward Pharm. Corp.*, 785 F.3d 625, 631 (Fed. Cir. 2015). GoDaddy contends that Express Mobile has failed to show that there is a genuine factual dispute on each of the above-listed elements. The Court disagrees.

On the first element—proof of underlying direct infringement—GoDaddy simply references the arguments it makes with respect to the direct infringement claims. For the reasons explained above, the Court overrules these arguments.

On the second and third elements—the knowledge elements—Express Mobile points to the e-mails and letters it sent to GoDaddy in 2013 and 2018. As discussed in the section concerning willfulness, these letters show a genuine dispute regarding whether GoDaddy (1) knew of the patents and (2) knew or should have known that the

induced acts constituted patent infringement. The 2013 e-mails informed GoDaddy of the existence of the asserted patents and contained claim charts related to WordPress, and the 2018 letter expressly accused GoDaddy of infringement.

On the last element, specific intent, the Court likewise finds that there is a genuine factual dispute. Express Mobile cites numerous documents and guides created and distributed by GoDaddy to its customers instructing them on how to use the accused products. These guides suggest that GoDaddy wanted its customers to use the accused products, including in ways that would infringe upon the asserted patents. A reasonable jury could thus conclude that GoDaddy possessed the specific intent for its customers to infringe the asserted patents.

## **ii. Contributory infringement**

"To establish contributory infringement, the patent owner must show the following elements . . . 1) [] there is direct infringement, 2) [] the accused infringer had knowledge of the patent, 3) [] the component has no substantial noninfringing uses, and 4) [] the component is a material part of the invention." *Fujitsu Ltd. v. Netgear Inc.*, 620 F.3d 1321, 1326 (Fed. Cir. 2010). The first two elements are identical to the first two elements of induced infringement. For the reasons provided above, the Court finds that a reasonable jury could find in favor of Express Mobile on these issues. GoDaddy does not dispute that there are genuine factual disputes regarding the last two elements. Accordingly, GoDaddy is not entitled to summary judgment.

## **2. Daubert motion**

GoDaddy requests that the Court exclude certain testimony from Dr. Kevin Almeroth concerning the technical importance of the claimed inventions and from Walter



Bratic concerning damages. The Court takes each in turn.

**a. Dr. Almeroth on the technical importance of the claimed inventions**

GoDaddy contends that the Court should exclude Dr. Almeroth's testimony regarding the technical importance of the claimed inventions for three reasons. None are persuasive. First, GoDaddy contends that the fact that Dr. Almeroth fails to use its non-infringing WSB V6 product as a baseline for comparison makes his opinions unreliable. Express Mobile disputes that WSB V6 is an acceptable non-infringing alternative that must be considered. The Court agrees. There are significant differences between the two products, meaning that it was not necessary for Dr. Almeroth to consider WSB V6. For example, WSB V6 is no longer being offered to new customers, and GoDaddy has made efforts to move customers off of the product. Additionally, testimony from Dr. Almeroth explains that WSB V6 lacks certain critical features of the accused product, including a WYSIWYG (what-you-see-is-what-you-get) website designer. Almeroth Opening Report ¶¶ 810–11; Almeroth Reply Report ¶ 526.

GoDaddy's second argument is that Dr. Almeroth's opinions are unreliable because he fails to consider GoDaddy's customer survey data. This is inaccurate. Dr. Almeroth states in his reply report that he "considered surveys and marketing materials to the extent they exist." Almeroth Reply Report ¶ 104. He also considered and cited internal defendant documents that evaluated customer preferences. Almeroth Opening Report ¶¶ 102–09.

Finally, GoDaddy argues that Dr. Almeroth fails to provide quantitative or mathematical support for his determinations. Again, this argument holds no water. Dr.

Almeroth thoroughly explains the basis for his opinion: he states that he determined the relative weights of the features based on how technical the feature is, whether any technical aspects are standard or conventional, and the extent to which technical difficulty and/or creativity is involved. Almeroth Opening ¶¶ 825–34. For these reasons, the Court declines to exclude Dr. Almeroth's testimony concerning the technical importance of the claimed inventions.

**b. Bratic on damages**

GoDaddy asks the Court to exclude Walter Bratic's opinions concerning damages for three reasons. First, GoDaddy contends that Bratic improperly uses Almeroth's technical weightings as the only basis for his royalty base calculations. This contention, however, is directly contradicted by Bratic's report, in which states that he looked at "the totality of evidence, including indicators of demand for the Website Builder as reflected in GoDaddy's market surveys." Bratic Report ¶ 184.

Second, GoDaddy contends that Bratic erred by using subscription count data rather than revenue data in his royalty calculations. Although it may be that GoDaddy's suggested calculation method is superior to Bratic's, this is not a basis for exclusion. GoDaddy presents numerous reasons to prefer calculations based on revenue data versus calculations based on subscription count data, but ultimately none suggest that a royalty calculation based on subscription count is so unreliable that it should be excluded. At trial, GoDaddy will have the opportunity to make its arguments to the jury, but for now, there is no basis to exclude the evidence.

Third, GoDaddy contends that the Court must exclude Bratic's opinions because his calculated royalty would be financially catastrophic. The Court disagrees. The

parties dispute GoDaddy's worldwide net revenue for the accused products and accordingly dispute whether Bratic's calculation royalty would be an unreasonable percentage of this amount. It is up to the jury to determine between these competing positions.

**D. Express Mobile's combined summary judgment/*Daubert* motion**

**1. Motion for summary judgment**

Express Mobile moves for summary judgment relating to certain invalidity and equitable defenses.

**a. Patent eligibility**

Express Mobile seeks summary judgment regarding whether the asserted patents are patent-eligible under the *Alice/Mayo* two-step framework. *See Alice Corp. Pty. Ltd. v. CLS Bank Int'l*, 573 U.S. 208 (2014); *Mayo Collaborative Servs. v. Prometheus Labs., Inc.*, 566 U.S. 66 (2012). As an initial matter, GoDaddy does not contest eligibility regarding the '397 and '168 patents, so the Court grants Express Mobile's motion with respect to these patents. GoDaddy does, however, contest the eligibility of the '755, '287, and '044 patents.

The first step of the *Alice/Mayo* two-step framework asks whether the asserted patent is directed to abstract ideas. *Alice*, 573 U.S. at 217. If it is not directed to abstract ideas, then the patent is valid. *Id.* If the patent is directed to abstract ideas, the court must ask if the patent claims inventive concepts. *Id.* If so, the patent is valid. *Id.* If not, the patent is invalid. *Id.*

**i. Step one**

Express Mobile contends the '755, '287, and '044 patents are not directed to

abstract ideas because they are directed to specific computing architecture that allows internet-connected devices to easily connect to and display the content of web services available on the internet. GoDaddy argues that the asserted patents are directed to the abstract idea of separating code into a device-independent aspect and a device-dependent aspect. The Court finds that the asserted patents are not directed to an abstract idea and thus that they are patent-eligible.

GoDaddy's interpretation takes much too simplistic a view of the asserted claims. It asserts that the patents are directed to an abstract idea because the segregation of device-independent and device-dependent code is the only new feature disclosed in the patents. But the first step of the *Alice/Mayo* framework does not ask what inventive concepts the patent claims—that inquiry is reserved for step two. *Id.* Instead, step one requires the court to look at "the claims as an ordered combination, without ignoring the requirements of the individual steps." *McRO, Inc. v. Bandai Namco Games Am. Inc.*, 837 F.3d 1299, 1313 (Fed. Cir. 2016). GoDaddy ignores certain parts of the claims because it contends they are "well known, routine, and conventional." Dkt. no. 189 at 30. This is not the proper analysis. Looking at the claims as an ordered combination, the Court finds that the patents are directed to specific computing architecture and not an abstract idea. It thus grants summary judgment in favor of Express Mobile on the issue of patent eligibility.

**ii. Step two**

Even if the '755, '287, and '044 patents were directed to an abstract idea, the Court would still grant summary judgment in favor of Express Mobile on patent eligibility because the patents claim inventive concepts. Specifically, the Court finds that these

patents claim a particular architecture that allows for the efficient display of web service content on a device platform.

GoDaddy argues that the patents do not claim inventive concepts because they merely claim implementation of an abstract idea (namely, separating code into a device-independent aspect and a device-dependent aspect) using generic computer components. In support of its contention, GoDaddy separates all of the individual aspects of the claim and concludes that all but one aspect was previously known in the art. The problem with this divide-and-conquer approach, however, is the patents' inventive concept is not each aspect of the claim individually. Rather, the inventive concept is the particular architecture of combining these components in a specific way to create an enhanced benefit. Summary judgment in favor of Express Mobile on patent eligibility is therefore warranted.

### **iii. Utility argument**

Lastly, Express Mobile seeks summary judgment on GoDaddy's argument that certain claims of the '755 patent "fail the requirements of 35 U.S.C. § 101 because the term 'where said Application is a device-dependent code' lacks utility." Dkt. no. 162, ex. 6 at 35. GoDaddy does not address this argument in its briefing and thus has waived or forfeited the point. See *Freeman v. Pittsburgh Glass Works, LLC*, 709 F.3d 240, 250 (3d Cir. 2013). The Court accordingly grants summary judgment in favor of Express Mobile on the § 101 defense.

### **b. Definiteness**

Section 112 requires that "a patent's claims, viewed in light of the specification and prosecution history, inform those skilled in the art about the scope of the invention

with reasonable certainty." *Nautilus, Inc. v. Biosig Instruments, Inc.*, 572 U.S. 898, 910 (2014). GoDaddy contends that certain of the asserted claims are indefinite in light of the language of the claims. The Court disagrees and grants summary judgment in favor of Express Mobile on this defense.

**i. "Data format type"**

GoDaddy asserted in its invalidity contentions that the claims of the '287 and '044 patents are indefinite because of the language "data format type of the symbolic name." Dkt. no. 162, ex. 6 at 38, 43. In the summary judgment briefing, however, GoDaddy acknowledged that the Court in *Shopify* already decided the issue in favor of Express Mobile, and it declined to provide any new arguments in support of its position. See *Shopify*, 2021 WL 4288113, at \*26. The Court thus grants summary judgment in favor of Express Mobile on the defense of indefiniteness regarding these patents.

**ii. "Data format class type"**

GoDaddy also asserts that the claim language "data format class type" that corresponds to a "subclass of User Interface (UI) objects" is indefinite. Dkt. no. 162, ex. 6 at 38, 43. As with "data format type," however, GoDaddy declined to provide any new arguments in its summary judgment briefing and instead incorporated Shopify's arguments to preserve the record. Accordingly, summary judgment in favor of Express Mobile on indefiniteness is granted for the same reasons given in *Shopify*. See *Shopify*, 2021 WL 4288113, at \*26.

**iii. "Utilizes information"**

GoDaddy also asserts that the language of the '044 patent stating that the Player "utilized information stored in said database to generate for the display of at least a

portion of said one or more web pages" is indefinite. Dkt. no. 162, ex. 6 at 42. Again, GoDaddy declined to make any new arguments in its summary judgment briefs in this case. Without additional argumentation, the Court finds it appropriate to decide the issue as Judge Andrews did in *Shopify*. See *Shopify*, 2021 WL 4288113, at \*27. Summary judgment is granted in favor of Express Mobile.

**iv. "Symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network"**

With respect to the '755 patent, GoDaddy contends that the claim term "symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network" is indefinite. Dkt. no. 162, ex. 6 at 33–34. Essentially, GoDaddy's argument is that "symbolic names" is unclear because the parties disagree over the scope of the claim: GoDaddy contends that the symbolic names relate only to UI components, whereas Express Mobile contends that symbolic names relate to web components generally. But GoDaddy provides no expert support for its interpretation. In its briefing, it merely cites to experts that criticize the terminology used by Express Mobile's expert. These experts, however, do not analyze the specific claim language or support GoDaddy's reading that "symbolic names" relate only to UI components. This is not enough to show a genuine dispute regarding whether a person of ordinary skill in the field would be able to understand what is claimed. Summary judgment in favor of Express Mobile is warranted.

**v. "UI Object corresponds to"**

GoDaddy contends that certain claims of the '755 and '287 patents are indefinite

based on the language "UI object corresponds to [the/a] web component included in said registry selected from the group consisting of an input of the web service and an output of the web service." *Id.* at 34, 38. Express Mobile disputes that the term is unintelligible and states that a person of ordinary skill in the field "would understand that this claim term . . . simply refers to UI objects on a display (such as text form fields, buttons, YouTube video controls, maps, etc.) that correspond to web components that relate to an input or output of a web service (such as a YouTube or Google Maps web service)." Dkt. no. 158 at 20. Again, GoDaddy's only support for its argument is its experts' testimony criticizing the terminology used by Express Mobile's expert. For the reasons given above, this is insufficient to create a genuine factual dispute, and thus the Court grants summary judgment in favor of Express Mobile.

**vi. "Authoring tool configured to"**

GoDaddy contends that the phrase "authoring tool configured to" in the '287 patent is a nonce word that is subject to 35 U.S.C. § 112, ¶ 6. Dkt. no. 162, ex. 6 at 38–39. The Court disagrees. When a claim does not use the word "means," there is a presumption that the claim language does not invoke § 112, ¶ 6. *Williamson*, 792 F.3d at 1348. Only where the term does not "have a sufficiently definite meaning as the name for structure" is this presumption overcome. *Id.* This is not the case here. "Authoring tool" was previously construed by the Court to mean "a system, with a graphical interface, for generating code to display content on a device screen." Dkt. no. 129 at 1. Contrary to GoDaddy's contention, the Court's construction references a sufficiently specific structure, so § 112, ¶ 6 does not apply.



**vii. "Said code"**

GoDaddy's last argument is that claim 11 of the '287 patent is indefinite because the term "said code" fails to inform a person skilled in the field about the scope of the invention. Dkt. no. 162, ex. 6 at 39. Specifically, it contends that "said code" is unclear because it could refer to either the Application code, the Player code, or both. The Court disagrees. The most natural reading of the term "said code," in context, is that the term refers to both the Application and the Player code. This is supported by Express Mobile's experts. See, e.g., Almeroth Opening Report ¶ 692. GoDaddy, on the other hand, fails to cite any expert testimony supporting its contention that the term is unclear. For these reasons, the Court grants summary judgment in favor of Express Mobile on the defense of indefiniteness on this claim.

**a. Equitable defenses**

Express Mobile seeks summary judgment on several equitable affirmative defenses that GoDaddy asserts in its answer to the amended complaint.

**i. Laches, patent exhaustion, express license, and disclaimer**

Express Mobile seeks summary judgment on GoDaddy's laches, patent exhaustion, express license, and disclaimer defenses. GoDaddy does not address these arguments in its response briefing and thus waives its response. See *Freeman, LLC*, 709 F.3d at 250. Express Mobile is entitled to summary judgment on these defenses.

**ii. Equitable estoppel, implied license, and acquiescence**

Express Mobile also seeks summary judgment on GoDaddy's equitable estoppel,

implied license, and acquiescence defenses. As an initial matter, the Court grants summary judgment in favor of Express Mobile on the implied license defense.

GoDaddy fails to make any argument specifically addressing this defense and instead relies on a single sentence in a footnote stating that summary judgment on implied license is improper where summary judgment on equitable estoppel is improper. See dkt. no. 189 at 6 n. 3. Given this lack of development, the Court finds that the argument is waived. See *N.J. Media Grp. Inc. v. United States*, 836 F.3d 421, 436 n. 20 (3d Cir. 2016) (determining that an argument was waived based on its “utterly undeveloped character”).

Under Federal Circuit precedent, a party raising a defense of equitable estoppel must prove the following three elements:

(1) The patentee, who usually must have knowledge of the true facts, communicates something in a misleading way, either by words, conduct or silence. (2) The alleged infringer relies upon that communication. (3) And the accused infringer would be harmed materially if the patentee is later permitted to assert any claim inconsistent with his earlier conduct.

*Vanderlande Indus. Nederland BV v. I.T.C.*, 366 F.3d 1311, 1324 (Fed. Cir. 2004) (citation omitted). The Court finds that there are genuine disputes of fact regarding each element precluding summary judgment.

On the first element, GoDaddy presents evidence relating to the 2013 discussions between the parties regarding the potential sale of the '397 and '168 patents. For example, when GoDaddy asked why Express Mobile thought it would be interested in the patents, Express Mobile responded that it thought GoDaddy would be interested because of the “general area in which the patents apply.” Dkt. no. 165-10 at XMO\_GD00129671. There was no suggestion that GoDaddy might be interested

because its products infringed Express Mobile's patents. *See generally id.* Additionally, the claim charts that Express Mobile sent to GoDaddy involved the WordPress product, not either of the accused products in this case. *Id.* at XMO\_GD00129668. And, even if GoDaddy should have suspected that Express Mobile believed that it had a claim for infringement, Express Mobile's delay in suing for five and one-half years reasonably suggested to GoDaddy that Express Mobile was not going to sue.

On the second element, GoDaddy has shown that there is a genuine factual dispute regarding whether it relied on Express Mobile's conduct. A reasonable jury could conclude that, had GoDaddy known that it faced legal liability, it would not have continued to pour money and time into the accused products without seeking a license. GoDaddy provides evidence that it sought licenses in other contexts, which supports its contention that it would have sought a license here.

The last element is whether the alleged infringer would be materially prejudiced if the patentee was allowed to proceed with its claim. Again, there is a genuine dispute of fact on this issue. GoDaddy presents evidence that it invested significant resources into the development of the WSB and MWP products, which is sufficient to permit a reasonable factfinder to find prejudice. For these reasons, Express Mobile is not entitled to summary judgment on GoDaddy's equitable estoppel defense.

Next, the Court concludes that Express Mobile is entitled to summary judgment on the acquiescence defense because it is duplicative of the equitable estoppel defense. *See State Contracting & Eng'g Corp. v. Condotte Am., Inc.*, 346 F.3d 1057, 1065 (Fed. Cir. 2003) (noting that the trial judge dismissed the equitable estoppel defense because "he deemed that defense duplicative of the acquiescence and consent

defense"). Acquiescence is a more general defense that includes defenses like equitable estoppel. See *Applications in Internet Time, LLC v. RPX Corp.*, 897 F.3d 1336, 1357 (Fed. Cir. 2018) ("Although acquiescence furnishes the most apt single label for these reasons, several distinctive principles can be identified [including equitable estoppel].") (internal citation and quotation marks omitted). Because the Court declines to grant summary judgment on the equitable estoppel defense, the acquiescence defense is superfluous.

### iii. Unclean hands

An unclean hands defense requires a showing of five elements: "(1) a party seeking affirmative relief (2) is guilty of conduct involving fraud, deceit, unconscionability, or bad faith (3) directly related to the matter in issue (4) that injures the other party (5) and affects the balance of equities between the litigants." *Sun Microsystems, Inc. v. Versata Enters.*, 630 F. Supp. 2d 395, 410 (D. Del. 2009). The parties dispute whether the second element requires a showing that the alleged conduct is "unconscionable" or that it would "shock the moral sensibilities of the judge." See *Honeywell Int'l, Inc. v. Universal Avionics Sys. Corp.*, 498 F. Supp. 2d 305, 310 (D. Del. 2005). The Court need not address this dispute, however, because it concludes that, even under GoDaddy's interpretation, no reasonable jury could find that Express Mobile acted with fraud, deceit, unconscionability, or bad faith.

According to GoDaddy, Express Mobile acted with unclean hands because it intentionally misled GoDaddy into thinking that it was not going to sue for patent infringement. GoDaddy argues that Express Mobile delayed suing GoDaddy to encourage it to invest more money into its products and continue to grow its business,

thus increasing any potential award from litigation or settlement. The record does not support GoDaddy's assertions, however. Although it is true that Express Mobile waited to sue GoDaddy, there is no evidence suggesting that its motivation for waiting was to increase its potential recovery. The evidence GoDaddy cites suggests that Express Mobile had targeted GoDaddy for litigation in 2014 but needed to delay suit in order to organize its litigation strategy, seek funding, and retain counsel. Without more, it would be unreasonable for a jury to conclude that Express Mobile acted with fraud, deceit, unconscionability, or bad faith. For these reasons, the Court grants summary judgment in favor of Express Mobile on the unclean hands defense.

## **2. *Daubert* motion**

Express Mobile contends that certain of David Perry's damages opinions should be excluded. First, Express Mobile argues that certain of Perry's opinions should be excluded because they are based on false circumstances that do not involve a willing licensor and licensee. Second, Express Mobile argues that certain of Perry's opinions rely on non-existent or misleading transactions. Lastly, Express Mobile argues that certain of Perry's opinions that rely on certain settlement agreements should be excluded. The Court disagrees with the first of these contentions but agrees with the other two.

### **a. False circumstances**

Express Mobile argues that certain of Perry's opinions should be excluded because they are based on three false circumstances that do not involve a willing licensor and willing licensee: (1) the GE agreement; (2) the RPX offer; and (3) the gross revenues of unlicensed companies.

**i. GE Agreement**

Express Mobile was represented by Green Espel for the patent auction.

According to Steve Rempell, Express Mobile's CEO, under the agreement between Express Mobile and Green Espel (the "GE agreement"), "Express Mobile could walk away from any offer less than 5 million." Dkt. no. 190-17 at 286:12-14. GoDaddy contends that this means that Express Mobile was bound to accept any offer at or above \$5 million, and some of Perry's damages opinions are based on this assumption. Express Mobile disputes that Rempell's testimony indicates that Express Mobile was required to accept an offer at or above \$5 million. Accordingly, it contends that the parts of Perry's testimony that relies on this "false circumstance" are unreliable and should be excluded.

The Court finds that the disputed testimony is ambiguous. Although Rempell stated that the GE Agreement authorized Express Mobile to walk away from any offer less than \$5 million, he does not, as GoDaddy contends, address whether Express Mobile was obligated to accept an offer at or above \$5 million. At the same time, Rempell does not state, as Express Mobile suggests, that \$5 million was the floor of what Express Mobile would have accepted.

Because the evidence is open to interpretation, both parties' positions are supportable, and Perry's calculations are not automatically unreliable for having interpreted the GE Agreement in the way that he did. At trial, Express Mobile will have the opportunity to present its interpretation to the jury, and the jury will be able to determine whether Perry's opinions are credible in light of the assumptions on which they are based.

Express Mobile makes one final argument with respect to the GE Agreement: it argues that "Mr. Perry's opinions as the Web Components should be excluded for the additional reason that those patents had not yet issued at the time of the patent auction." Dkt. no. 158 at 35 n. 8. GoDaddy does not address this argument, and the Court finds it persuasive. Any opinion by Perry regarding the Web Component Patents based on the GE Agreement is excluded.

**ii. RPX Offer**

RPX is a patent aggregator that buys patents as a defensive measure to protect its member organizations from suit. During the patent negotiations, RPX invited Express Mobile to make an offer at \$3.5 million for the Web Design Patents and cross-licenses. Some of Perry's opinions are based on this offer (to make an offer).

Express Mobile argues that it was improper for Perry to consider the RPX offer because it does not constitute an agreement and as such does not reflect an agreement between a willing licensor and willing licensee. Additionally, Express Mobile emphasizes that the offer was not just for the Web Design Patents: RPX suggested that it would only agree to a \$3.5 million offer if cross-licenses were included. Thus, Express Mobile contends, the RPX offer cannot be reliably used to determine the value of the Web Design Patents alone.

The Court overrules this argument. Even if an offer was never made and the agreement never finalized, the invitation to make an offer provides relevant information regarding the perceived value of the patents in question. Specifically, it suggests that the maximum value of the accused patents (as estimated by a third-party buyer) is around \$3.5 million. Express Mobile's argument regarding the cross-licenses is similarly

not a reason to exclude Perry's testimony. These arguments go to the weight of this evidence, not its admissibility. At trial, Express Mobile may attempt to show that the RPX Offer should not be used to calculate a hypothetical license agreement between itself and GoDaddy. At this stage, however, there is no proper basis for the Court to exclude the evidence.

**iii. Gross revenues of unlicensed companies**

Express Mobile also argues that some of Perry's opinions should be excluded because they rely on the gross revenues of unlicensed companies and that this does not provide a value for the patented technology. GoDaddy contends that Perry relies on gross revenue, not to estimate the value of the patented technology, but instead to compare the relative sizes of different companies in accordance with Express Mobile's licensing program. Express Mobile's licensing program uses the comparative sizes of companies (based on total annual revenues) to determine how much each company should pay to license the patents.

Express Mobile fails to address GoDaddy's argument, and the Court finds it a persuasive one. Because Perry is not using the gross revenues to directly calculate the value of a hypothetical license, it is irrelevant whether that gross revenue accounts for all the revenue derived from the patented technology. For this reason, the Court declines to exclude Perry's opinions on this basis.

**b. Non-existent or misleading transactions**

Express Mobile contends that certain of Perry's opinions that rely on non-existent or misleading transactions should be excluded. The Court agrees.



**i. Bankruptcy filing evidence**

Perry discusses two pieces of evidence from Rempell's bankruptcy. The first is an e-mail from Rempell's attorney stating that his Express Mobile stock was probably worthless. The second is a bankruptcy filing stating that Rempell's sixty-five percent stake in Express Mobile was worth \$6,852, thus suggesting an overall value of \$10,541.

Express Mobile contends that the bankruptcy evidence is not relevant because the value of the Express Mobile stock did not include the value of the patents. This is supported by Rempell's deposition testimony, see dkt. no. 162-28 at 313-316, and GoDaddy does not address this point. The Court thus excludes Perry's opinions that are based on this evidence.

**ii. CY Digital license agreement**

Perry also considers an agreement between Express Mobile and CY Digital in his opinions. Express Mobile argues that Perry's reliance on the agreement is improper because the agreement was an agreement to invest in CY Digital, not a licensing agreement as GoDaddy so contends. Express Mobile concedes that it and CY Digital discussed the possibility of executing a licensing agreement, but that agreement was never finalized. As such, Express Mobile argues that the draft agreement cannot be the basis for Perry's damages opinions.

The Court agrees. Nothing in the record supports the existence of a licensing agreement. GoDaddy cites to language in a draft document, but in the final draft, that language was removed. Dkt. no. 162-30 at 145–47. Even Perry himself acknowledges that the draft license agreement was never executed: "CY Digital prepared a document marked draft stating that it had been granted rights to use the Patents-In-Suit. Express

Mobile states that the agreement with CY Digital was terminated on November 11, 2019." Aug. 31, 2021 Perry Report at 30–31. Without a finalized agreement, it is difficult to draw meaning from the terms of the draft agreement. There is no evidence that the draft agreement actually represented either party's perceived value of a licensing agreement. In particular, it is unclear from the record whether either party offered the terms in the draft agreement to the other or how far along in the negotiating process the parties were when the potential deal was scrapped. Without more, it is unreasonable for Perry to rely on the draft agreement as a basis for his opinions. For this reason, the Court agrees that Perry's opinions arising from this draft agreement should be excluded.

**c. Non-comparable settlement agreements**

Express Mobile entered into fifty-five settlement agreements with companies that it sued. See Perry Report at 22–28. In its decision in *Shopify*, the Court ruled that these settlement agreements may be relied on to support opinions regarding the appropriate form of royalties but not the appropriate amount of royalties. *Shopify*, 2021 WL 4288113, at \*28. Express Mobile argues that Perry's opinions run afoul of this ruling by using the settlement agreements to support opinions regarding how much the royalty payment should be.

The Court agrees. Although Perry does use the settlement agreements to argue for a lump sum form of royalty, he also uses the information to make opinions about the proper size of the royalty payment:

Mr. Bratic opines that a hypothetical negotiation between Express Mobile and GoDaddy for a non-exclusive license to the Patents-In-Suit would have resulted in a running royalty structure involving total royalty payments of \$346 million through April 2021. Mr. Bratic's opinion of the

royalties due from one of many alleged infringers represents 1,978 times the largest single lump-sum amount of \$175,000 that Express Mobile has received in any of the 55 agreements.

Perry Report at 28. Settlement agreements used to determine the proper size of a royalty payment must be comparable to a hypothetical agreement between the parties. See *LaserDynamics, Inc. v. Quanta Computer, Inc.*, 694 F.3d 51, 79 (Fed. Cir. 2012) ("When relying on licenses to prove a reasonable royalty, alleging a loose or vague comparability between different technologies or licenses does not suffice."). GoDaddy concedes that Perry did not perform a comparability analysis. The Court thus excludes this testimony.

### **Conclusion**

The disputed claim terms are construed in accordance with the conclusions set forth in this Memorandum Opinion and Order. Additionally, the Court denies both parties' motions to strike [182] [209]. With respect to the parties' combined summary judgment/*Daubert* motions [156] [157], the Court grants the motions in part and denies the motions in part. The Court grants summary judgment in favor of GoDaddy on the issue of infringement regarding the '397 and '168 patent claims but otherwise denies GoDaddy's motion for summary judgment. The Court grants summary judgment in favor of Express Mobile on GoDaddy's defenses of patent eligibility, utility, indefiniteness, laches, patent exhaustion, express license, disclaimer, implied license, acquiescence, and unclean hands but otherwise denies Express Mobile's motion for summary judgment. Lastly, the Court excludes the portion of David Perry's testimony regarding the Web Component Patents that is based on the GE Agreement and the portion of his testimony that is based on the Rempell bankruptcy evidence, the CY

Digital Agreement, and the settlement agreements but otherwise denies the *Daubert* motions. The parties are directed to file by August 15, 2022 a joint status report addressing any scheduling issues going forward and reporting on the history and current status of any settlement discussions. The case is set for a telephonic status hearing on August 17, 2022 at 8:40 a.m. CT (9:40 a.m. ET), using call-in number 888-684-8852, access code 746-1053. The Court reserves the right to vacate the status hearing if it determines from the status report that a hearing is not needed.

  
MATTHEW F. KENNELLY  
United States District Judge

Date: August 8, 2022

**Subject: Activity in Case 1:19-cv-01937-MFK-JLH Express Mobile, Inc. v. GoDaddy.com, LLC Oral Order**

**This is an automatic e-mail message generated by the CM/ECF system. Please DO NOT RESPOND to this e-mail because the mail box is unattended.**

**\*\*\*NOTE TO PUBLIC ACCESS USERS\*\*\*** Judicial Conference of the United States policy permits attorneys of record and parties in a case (including pro se litigants) to receive one free electronic copy of all documents filed electronically, if receipt is required by law or directed by the filer. PACER access fees apply to all other users. To avoid later charges, download a copy of each document during this first viewing. However, if the referenced document is a transcript, the free copy and 30 page limit do not apply.

**U.S. District Court**

**District of Delaware**

**Notice of Electronic Filing**

The following transaction was entered on 10/27/2022 at 4:33 PM EDT and filed on 10/27/2022

**Case Name:** Express Mobile, Inc. v. GoDaddy.com, LLC

**Case Number:** [1:19-cv-01937-MFK-JLH](#)

**Filer:**

**Document Number:** 271(No document attached)

**Docket Text:**

**ORAL ORDER: The Court denies plaintiff's motion for reargument [262]. Certain of the points in the motion are made for the first time in seeking reconsideration and are thus waived. Other points made in the motion are simply rehashes of points the Court considered and rejected. This likewise is not an appropriate basis for reconsideration. All of that aside the Court has considered the points made by plaintiff and finds them lacking in merit. The Court is not persuaded that it made any clear or manifest errors in its August 8 decision. Ordered by Judge Matthew F Kennelly on 10/27/2022. (smg)**

**1:19-cv-01937-MFK-JLH Notice has been electronically mailed to:**

Beth Moskow-Schnoll moskowschnollb@ballardspahr.com, LitDocket\_East@ballardspahr.com, harmonw@ballardspahr.com

Timothy Devlin tdevlin@devlinlawfirm.com, dlflitparas@devlinlawfirm.com, mmclain@devlinlawfirm.com

John L. Abramic jabramic@steptoe.com, jcooper@steptoe.com

Tron Y. Fu tfu@steptoe.com, lthompson@steptoe.com

Brian W. LaCorte lacorteb@ballardspahr.com

Robert F. Kappers rkappers@steptoe.com, #-FirmPSDocketing@steptoe.com

Brittany M. Giusini giusinib@ballardspahr.com, LitDocket\_East@ballardspahr.com, harmonw@ballardspahr.com

Katherine H. Johnson kjohnson@steptoe.com, lthompson@steptoe.com

James R. Nuttall jnuttall@steptoe.com, jcooper@steptoe.com

Christopher A. Suarez csuarez@steptoe.com, pbirdsall@steptoe.com

Jonathon A. Talcott talcottj@ballardspahr.com, LitDocket\_West@ballardspahr.com

Brian S.S. Auerbach auerbachb@ballardspahr.com, LitDocket\_West@ballardspahr.com

Michael Dockterman mdockterman@steptoe.com, meckstein@steptoe.com

Shaton C. Menzie menzieS@ballardspahr.com, louisB@ballardspahr.com

Andrew M. Hensley hensleya@ballardspahr.com, LitDocket\_West@ballardspahr.com

**1:19-cv-01937-MFK-JLH Filer will deliver document by other means to:**

**IN THE UNITED STATES DISTRICT COURT  
FOR THE DISTRICT OF DELAWARE**

_____	)	
EXPRESS MOBILE, INC.,	)	
	)	
Plaintiff,	)	
	)	
v.	)	Civil Action No.1:19-cv-01937-MFK
	)	
GODADDY.COM, LLC,	)	
	)	
Defendant.	)	
_____	)	

**FINAL JURY INSTRUCTIONS**

**TABLE OF CONTENTS**

1.	INTRODUCTION .....	1
2.	THE PARTIES AND THEIR CONTENTIONS .....	2
3.	THE PATENT CLAIMS .....	3
3.1.	THE ROLE OF CLAIMS IN THE PATENT .....	3
3.2.	INDEPENDENT AND DEPENDENT CLAIMS .....	4
3.3.	CLAIM INTERPRETATION.....	5
4.	INFRINGEMENT.....	7
4.1.	INFRINGEMENT GENERALLY.....	7
4.2.	DIRECT INFRINGEMENT BY "LITERAL INFRINGEMENT" .....	8
4.3.	DIRECT INFRINGEMENT – ONE OR MORE SYSTEM COMPONENTS LOCATED OUTSIDE THE UNITED STATES .....	9
4.4.	WILLFUL INFRINGEMENT .....	10
5.	INVALIDITY .....	11
5.1.	PRIOR ART .....	12
5.2.	OBVIOUSNESS .....	13
5.3.	OBVIOUSNESS – ADDITIONAL FACTORS INDICATING NONOBVIOUSNESS .....	14
5.4.	OBVIOUSNESS — COMBINING OF PRIOR ART REFERENCES.....	15
6.	DAMAGES.....	16
6.1.	DATE OF COMMENCEMENT OF DAMAGES .....	17
6.2.	REASONABLE ROYALTY—DEFINITION .....	18
6.3.	DAMAGES – LUMP SUM VS. RUNNING ROYALTY .....	19
6.4.	REASONABLE ROYALTY—RELEVANT FACTORS.....	20
7.	DELIBERATION AND VERDICT .....	22
7.1.	INTRODUCTION .....	22
7.2.	UNANIMOUS VERDICT.....	23



7.3.	DUTY TO DELIBERATE .....	24
7.4.	COURT HAS NO OPINION.....	25

**1. INTRODUCTION**

Members of the jury, it is time for me to instruct you about the law that you must follow in deciding this case.

Each of you has been provided a copy of these instructions. If you prefer, you may read along as I deliver them. You will be able to take your copies with you into your deliberations and refer to them at any time, if necessary.

It is your duty as jurors to follow the law as I will state it to you, and to apply that law to the facts as you find them from the evidence in the case. You may not single out one instruction alone as stating the law but must consider the instructions as a whole. The instructions as a whole include not only these instructions and the preliminary instructions, but also any instructions I gave at various points during trial. You should not be concerned with the wisdom of any rule that I state. Regardless of any opinion that you may have as to what the law may be-or should be-it would violate your sworn duty to base a verdict upon any view of the law other than that which I will now give you.

In deciding what the facts are, you may have to decide what testimony you believe and what testimony you do not believe. The weight of the evidence to prove a fact does not necessarily depend on the number of witnesses who testify. What is more important is how believable the witnesses were, and how much weight you think their testimony deserves.

**2. THE PARTIES AND THEIR CONTENTIONS**

As I previously told you, Express Mobile alleges that certain GoDaddy products infringe the Asserted Claims of the Asserted Patents.

GoDaddy denies that it has infringed the Asserted Claims of the Asserted Patents and also argues that the Asserted Claims are invalid.

In this case, you must decide the issues according to the instructions I give you. In general, these issues are:

Whether Express Mobile has proven by a preponderance of the evidence that GoDaddy has infringed any of the Asserted Claims of the Asserted Patents.

Whether Express Mobile has proven by a preponderance of the evidence that GoDaddy has willfully infringed any of the Asserted Claims of the Asserted Patents.

Whether GoDaddy has proven by clear and convincing evidence that any of the Asserted Claims of the Asserted Patents are invalid.

If you find that any Asserted Claim of any of the Asserted Patents is infringed and is not invalid, then you must decide what damages has Express Mobile proven by a preponderance of the evidence that it is entitled to compensate Express Mobile for GoDaddy's infringement.

Your decision will be recorded in a verdict form that I will provide to you.

### **3. THE PATENT CLAIMS**

#### **3.1. THE ROLE OF CLAIMS IN THE PATENT**

Before you can decide the issues in this case, you will need to understand the role of patent "claims." Patent claims are the numbered sentences at the end of each patent. The claims are important because the words in those sentences define what a patent covers. The rest of the patent includes figures and text that provide a description and examples of the invention and context for the claims, but it is the claims that define the breadth of the patent's coverage. Therefore, what a patent covers depends, in turn, on what each of its claims cover.

A claim states, in words, a set of requirements that describe what the claim covers. Each claim sets forth its requirements in a single sentence. The requirements of a claim are often referred to as "claim elements" or "claim limitations." The scope of a patent is assessed claim-by-claim. When a thing (such as a product or a method) meets all the requirements of a claim, the claim is said to "cover" that thing, and that thing is said to "fall" within the scope of that claim. In other words, a claim covers a product or method when each of the claim elements or limitations is present in that product or method.

### **3.2. INDEPENDENT AND DEPENDENT CLAIMS**

This case involves two types of patent claims: independent claims and dependent claims.

An "independent claim" describes all the requirements that must be met in order to be covered by that claim. Thus, it is not necessary to look at any other claim to determine what an independent claim covers. In this case, Asserted Claims 1 and 12 of the '755 patent, claim 1 of the '287 patent, and claim 1 of the '044 patent are independent claims. The remaining Asserted Claims, claims 3, 16, and 22 of the '755 patent, claim 13 of the '287 patent and claims 11, 13, 17 and 19 of the '044 patent, are "dependent claims."

A "dependent claim" does not by itself describe all the requirements of the claim but refers to another claim for some of its requirements. In this way, the claim "depends" on another claim.

A dependent claim incorporates all the requirements of the claim(s) that it depends on. The dependent claim then adds its own additional requirements. To determine what a dependent claim covers, it is necessary to look at both the dependent claim and any other claim that it depends on. Here, for example, claim 3 of the '755 patent depends on claim 1. A product must meet all the requirements of both dependent claim 3 and independent claim 1, in order for you to find that it is covered by dependent claim 3.

### **3.3. CLAIM INTERPRETATION**

It is my job as the judge to determine the meaning of any claim language from these patents that needs interpretation. You must accept the meanings that I give you and use them when you decide whether any claim is infringed or invalid.

In order to decide whether or not each claim is invalid and whether it has been infringed, you will first need to understand what each claim covers. The first step is to understand the meaning of the words used in the patent claim. Sometimes, the words in a patent claim are difficult to understand, and it is therefore difficult to understand what requirements these words impose. The law says that it is my role to define the terms of the claims, and it is your role to apply my definitions to the issues that you are asked to decide in this case. Therefore, as I explained to you at the start of the case, I have determined the meaning of certain claim terms and I will provide you with those definitions. You must accept my definitions of these words as correct.

For any words in the claim for which I have not provided you with a definition, you should apply its ordinary and accustomed meaning as understood by a person of ordinary skill in the art. The parties agree that the definition of a person of ordinary skill in the art that you should apply in this case is someone who had a bachelors or graduate degree in computer science, mathematics, engineering, or a similar discipline, or equivalent practical experience, together with knowledge of software development and web design and development with graphical user interfaces and systems, together with approximately two or three years of experience in the field relating to web design and development. You should not take my definition of the language of the claims as an indication that I have a view regarding how you should decide the issues that you are being asked to decide, such as infringement or invalidity. These issues are yours to decide. I will now read you my interpretation of various claim terms and phrases:

<b>Claim Term</b>	<b>Patents</b>	<b>Construction</b>
application	'755, '287 and '044 patents	device-independent code containing instructions for a device and which is separate from the Player
player	'755, '287 and '044 patents	device-specific code which contains instructions of a device and which is separate from the Application
device-dependent code	'755 and '287 patents	code that is specific to the operating system, programming language, or platform of a device
device-independent code	'755 and '287 patents	code that is not specific to the operating system, programming language, or platform of a device
authoring tool / authoring tool configured to	'755, '287 and '044 patents	a system, with a graphical interface, for generating code to display content on a device screen
where said application is a device-dependent code	'755 patent	where said Application is a device-independent code
for evoking one or more web components	'755, '287 and '044 patents	for calling up one or more web components
registry	'755 and '287 patents	a database that is used for computing functionality
web component	'755, '287 and '044 patents	software objects that provide functionalities of a web service
web service	'755, '287 and '044 patents	A software system that supports interaction between devices over a network
symbolic name(s)	'755, '287 and '044 patents	No construction necessary

#### **4. INFRINGEMENT**

##### **4.1. INFRINGEMENT GENERALLY**

In this case, Express Mobile alleges that GoDaddy infringes claims 1, 3, 12, 16, and 22 of the '755 patent, claims 1 and 13 of the '287 patent, and claims 1, 11, 13, 17, and 19 of the '044 patent. I will refer to these as the "Asserted Claims."

You must determine the issue of infringement separately for each Asserted Claim. Infringement is assessed on a claim-by-claim basis. Therefore, there may be infringement of one claim but no infringement of another claim.

I will now instruct you how to decide whether Express Mobile has proven that GoDaddy has infringed the Asserted Claims. Express Mobile must prove infringement by a "preponderance of the evidence." That means Express Mobile has to produce evidence which, when considered in the light of all the facts, leads you to believe that the particular proposition you are considering is more likely true than not.

You have heard evidence about both Express Mobile's product development and GoDaddy's accused products. However, in deciding the issue of infringement you may not compare GoDaddy's accused products to Express Mobile's product development. Rather, you must compare the GoDaddy's accused product to the claims of the '755, '287, and '044 patents when making your decision regarding infringement.



#### **4.2. DIRECT INFRINGEMENT BY "LITERAL INFRINGEMENT"**

To prove infringement, Express Mobile must prove by a preponderance of the evidence that GoDaddy made, used, sold or offered for sale within the United States a product or method that meets all of the requirements of the particular patent claim you are considering.

To determine infringement, you must compare the accused products with each Asserted Claim. In making that comparison, you must use my claim definitions, along with the ordinary meaning in the field of the patent for the other terms in the claims. You must compare the accused products with each and every one of the requirements of a claim to determine whether all of the requirements of that claim are met by one or more of the accused products. If the accused product does not literally contain one or more elements recited in a claim, then you must find that the accused product does not directly infringe that claim.

**4.3. DIRECT INFRINGEMENT – ONE OR MORE SYSTEM COMPONENTS  
LOCATED OUTSIDE THE UNITED STATES**

Direct infringement requires that the accused system include every element listed in the claim.

Express Mobile claims that infringement occurred within the United States even though GoDaddy contends that some (but not all) of the elements of the claim were located outside of the United States. For infringement to occur within the United States, Express Mobile must prove, by a preponderance of the evidence, that the control of the system took place, and the benefit of the system was enjoyed, in the United States.

#### **4.4. WILLFUL INFRINGEMENT**

Express Mobile argues that GoDaddy willfully infringed Express Mobile's patents from October 22, 2019 through the present. If you have decided that GoDaddy has infringed, you must also address the additional issue of whether or not this infringement was willful. This requires you to determine whether Express Mobile proved that it is more likely than not that GoDaddy knew of Express Mobile's patents and infringed them deliberately or intentionally.

To determine whether GoDaddy acted willfully, consider all of the facts and assess what knowledge GoDaddy had at the time of the challenged conduct. Facts that you may consider include, but are not limited, to:

(1) Whether or not GoDaddy acted consistently with the standards of behavior for its industry;

(2) Whether or not GoDaddy intentionally copied a product of Express Mobile that is covered by the Asserted Patents;

(3) Whether or not GoDaddy reasonably believed it did not infringe or that the Asserted Patents were invalid;

(4) Whether or not GoDaddy made a good-faith effort to avoid infringing the Asserted Patents, for example, whether GoDaddy attempted to design around the Asserted Patents; and

(5) Whether or not GoDaddy tried to cover up its infringement.

**5. INVALIDITY**

I will now instruct you on the rules you must follow in deciding whether GoDaddy has proven that the Asserted Claims are invalid.

The law presumes the Asserted Claims are valid. For this reason, GoDaddy has the burden of proving invalidity by clear and convincing evidence. Proof by clear and convincing evidence consists of proof that the truth of a factual contention is highly probable. This is a higher burden than proof by a preponderance of the evidence.

GoDaddy contends that the Asserted Claims are invalid based on obviousness. The term "obviousness" has a special meaning under patent law, which I will explain to you in these instructions.

In making your determination as to invalidity, you should consider each claim separately.

### **5.1. PRIOR ART**

In addressing Defendant's invalidity defenses, you will have to consider what is disclosed in the "prior art." That which came before the invention of the patents is referred to as "prior art."

The parties agree that the following references are prior art:

BlackBerry MDS in combination with the knowledge and routine skill of one of ordinary skill in the art is prior art to the Asserted Patents.

## **5.2. OBVIOUSNESS**

Defendant contends that the Asserted Claims of the '755, '287, and '044 patents are invalid because they are obvious.

In order to show that the claimed invention is obvious, Defendant must prove by clear and convincing evidence that a person of ordinary skill in the art of the invention, who knew about all the prior art existing at the time the invention was made would have conceived the invention at that time. Obviousness may be shown by considering one or more items of prior art in combination.

In deciding obviousness, you should put yourself in the position of a person with ordinary skill in the field at the time the invention was made. You must not use hindsight; in other words, you may not consider what is known now or what was learned from Plaintiff's patent. In addition, you may not use Plaintiff's patent as a roadmap for selecting and combining items of prior art.

In making your decision regarding obviousness, you are to consider each of the following factors:

1. The scope and content of the prior art. You may consider prior art that was reasonably relevant to the problem the inventor faced that a person of ordinary skill would consider in attempting to solve the problem.

2. Any differences between the prior art and the invention in the patent claim that you are considering.

3. The level of ordinary skill in the field of the invention at the time the invention was made.

4. Additional factors, if any, that indicate that the invention was obvious or not obvious. I will define them in the next instruction.

**5.3. OBVIOUSNESS – ADDITIONAL FACTORS INDICATING  
NONOBVIOUSNESS**

As I stated in the previous instruction, in deciding obviousness you should consider whether any of the following are true, which if so may indicate the invention was not obvious.

1. Has the invention achieved commercial success? If so, is that success based on the invention itself, rather than on advertising, promotion, sales tactics, or features of the product other than those found in the claimed invention?

2. Did others seek or obtain a license to the Patent(s)-in-Suit?

Not all of these factors may be present. No single factor is more or less important than the others.

#### **5.4. OBVIOUSNESS — COMBINING OF PRIOR ART REFERENCES**

The fact that each of the elements of the claim may be found in prior art references, if combined is not enough, by itself, to prove obviousness. In determining whether Defendant has proved obviousness, you may combine multiple items of prior art only if a person who has ordinary skill in the field would have been motivated to combine them when trying to solve the problem that is addressed by the claimed invention and would have had a reasonable expectation of success in doing so. In deciding this, you may consider, among other things, any of the following factors:

1. What the prior art suggests about combining;
2. The knowledge possessed by persons who have ordinary skill in the field of the invention; and
3. The effects of market pressures and design needs that existed at the time, and the number of identified and predictable solutions for those demands.



**6. DAMAGES**

If you find that GoDaddy infringed any valid claim of the asserted patents, you must then consider what amount of damages to award to Express Mobile. If you find that GoDaddy has not infringed any valid claim of the asserted patents, then Express Mobile is not entitled to any damages.

The fact that I am instructing you on damages does not mean that I believe that one party or the other should win in this case. My instructions about damages are for your guidance only in the event you find in favor of Express Mobile.

The damages you award must be adequate to compensate Express Mobile for any infringement. They are not meant to punish an infringer. Express Mobile has the burden to establish the amount of its damages by a preponderance of the evidence. While Express Mobile is not required to prove the amount of its damages with mathematical precision, it must prove them with reasonable certainty. You may not award damages that are speculative or based on guesswork.

Express Mobile seeks to recover a "reasonable royalty." A reasonable royalty is defined as the dollar amount Express Mobile and GoDaddy would have agreed upon in a hypothetical negotiation, which I will explain below, as a fee for GoDaddy's use of the claimed invention at the time just prior to when the alleged infringement began.

**6.1. DATE OF COMMENCEMENT OF DAMAGES**

In determining the amount of damages, you must determine when the damages began. In this case, damages, if any, began on June 23, 2015 for the '755 patent; on October 18, 2016 for the '287 patent; and on March 27, 2018 for the '044 patent.

## **6.2. REASONABLE ROYALTY—DEFINITION**

A royalty is a payment made to a patent holder in exchange for the right to make, use, or sell the claimed invention. A reasonable royalty is the amount of royalty payment that a patent holder and the alleged infringer would have agreed to in a hypothetical negotiation taking place at a time prior to when the infringement first began. In considering this hypothetical negotiation, you should focus on what the expectations of the patent holder and the alleged infringer would have been if they had entered into an agreement at that time, and if they had acted reasonably in their negotiations. In determining this, you must assume that both parties believed the patent was valid and infringed and that both parties were willing to enter into an agreement. The reasonable royalty you determine must be a royalty that would have resulted from the hypothetical negotiation, and not simply a royalty either party would have preferred. If evidence of things that happened after the infringement first began will aid you in assessing what royalty would have resulted from a hypothetical negotiation just prior to the first infringement, then you may consider such evidence.

### **6.3. DAMAGES – LUMP SUM VS. RUNNING ROYALTY**

A reasonable royalty can be paid either in the form of a one-time lump sum payment or as a "running royalty." Either method is designed to compensate the patent holder based on the infringer's use of the patented technology. It is up to you to determine based on the evidence what type of royalty, if any, is appropriate in this case.

#### **6.4. REASONABLE ROYALTY—RELEVANT FACTORS**

In determining the reasonable royalty, you should consider all the facts known and available to the parties at the time the infringement began. Some factors that you may consider are:

(1) The royalties received by the patent holder for the licensing of the Asserted Patents, proving or tending to prove an established royalty.

(2) The rates paid by the licensee for the use of other patents comparable to the Asserted Patents.

(3) The nature and scope of the license (for example, as exclusive or nonexclusive, or as restricted or non-restricted in terms of territory or with respect to whom the manufactured product may be sold).

(4) The licensor's established policy and marketing program to maintain his or her patent monopoly (for example, by not licensing others to use the invention, or by granting licenses under special conditions designed to preserve that monopoly).

(5) The commercial relationship between the licensor and licensee, such as whether they are competitors in the same territory and line of business, or whether they are inventor and promoter.

(6) The effect of derivative or convoyed sales (for example, the effect of selling the patented specialty in promoting sales of other products of the licensee, or the existing value of the invention to the licensor as a generator of sales of his non-patented items.

(7) The duration of the patent and the term of the license.

(8) The established profitability of the product made under the patents, its commercial success, and its current popularity.

(9) The utility and advantages of the patented property over the prior art, if any, that had

been used to achieve similar results.

(10) The nature of the patented invention, the character of commercial embodiments of it as owned and produced by the licensor, and the benefits to those who have used the invention.

(11) The extent to which the infringer has made use of the invention and any evidence that substantiates the value of that use.

(12) The portion of the profit, or of the selling price, that may be customary in the particular business or in comparable business to allow for the use of the invention or analogous inventions.

(13) The portion of the realizable profits that are due to the patented invention as distinguished from non-patented elements, the manufacturing process, business risks, or significant features or improvements added by the alleged infringer.

(14) The opinion testimony of qualified experts.

(15) The amount that a prudent licensee—who desired, as a business proposition, to obtain a license to manufacture and sell a particular article embodying the patented invention—would have been willing to pay as a royalty and yet be able to make a reasonable profit and that would have been acceptable to a prudent patent holder who was willing to grant a license.

No one factor is dispositive, and you can and should consider the evidence that has been presented to you in this case on each of these factors. You may also consider any other factors which in your mind would have increased or decreased the royalty the alleged infringer would have been willing to pay, and the patent holder would have been willing to accept, if they were acting as normally prudent business people.

## **7. DELIBERATION AND VERDICT**

### **7.1. INTRODUCTION**

That concludes the part of my instructions explaining the rules for considering some of the testimony and evidence. After you hear closing arguments of counsel, you will return to the jury room to begin your deliberations. Now, let me finish up by explaining some things about your deliberations in the jury room, and your possible verdicts.

Once you are all in the jury room, the first thing you should do is choose a presiding juror. The presiding juror should see to it that your discussions are carried on in an organized way and that everyone has a fair chance to be heard. You may discuss the case only when all jurors are present.

Once you start deliberating, do not talk to the jury officer, to me, or to anyone else except each other about the case. If you have any questions or messages, you must write them down on a piece of paper, sign them, and then give them to the jury officer. The officer will give them to me, and I will respond as soon as I can. I may have to talk to the lawyers about what you have asked, so it may take me some time to get back to you. You may continue to deliberate while you wait for me to answer. Any questions or messages normally should be sent to me through your foreperson.

## **7.2. UNANIMOUS VERDICT**

Your verdict must represent the considered judgment of each juror. Your verdict must be unanimous.

A verdict form has been prepared for you. The verdict form asks you a series of questions about the parties' claims and defenses.

(Go over the verdict form.)

You will take this form to the jury room. When you have reached unanimous agreement, your presiding juror will fill in and date the verdict form, and each of you will sign it. Unless you are directed otherwise in the verdict form, you must answer all of the questions asked, and you must all agree on each answer.

Advise the jury officer once you have reached a verdict. When you come back to the courtroom, you will hand the verdict to the jury officer, and I will read the verdict aloud.



### **7.3. DUTY TO DELIBERATE**

Now that all the evidence is in and the arguments are completed, you are free to talk about the case in the jury room. In fact, it is your duty to talk with each other about the evidence and to make every reasonable effort you can to reach a unanimous agreement. Talk with each other, listen carefully and respectfully to each other's views, and keep an open mind as you listen to what your fellow jurors have to say. Try your best to work out your differences. Do not hesitate to change your mind if you are convinced that other jurors are right and your original position was wrong. But do not ever change your mind just because other jurors see things differently, or just to get the case over with. In the end, your vote must be exactly that - your own vote. It is important for you to reach unanimous agreement, but only if you can do so honestly and in good conscience.

No one will be allowed to hear your discussions in the jury room, and no record will be made of what you say. So you should all feel free to speak your minds.

Listen carefully to what the other jurors have to say, and then decide for yourself.

**7.4. COURT HAS NO OPINION**

Let me finish up by repeating something that I said to you earlier. Nothing that I have said or done during this trial was meant to influence your decision in any way. You must decide the case yourselves based on the evidence presented.

Finally, if I have read any of these instructions inconsistently with the written text, you are to rely on the written instructions in your deliberations.

**IN THE UNITED STATES DISTRICT COURT  
FOR THE DISTRICT OF DELAWARE**

_____	)	
EXPRESS MOBILE, INC.,	)	
	)	
Plaintiff,	)	
	)	
v.	)	Civil Action No.1:19-cv-01937-MFK
	)	
GODADDY.COM, LLC,	)	
	)	
Defendant.	)	
_____	)	

**VERDICT FORM**

**INSTRUCTIONS:**

When answering the following questions and filling out this Verdict Form, please follow the directions provided throughout this Verdict Form. Your answer to each question must be unanimous. Some of the questions contain legal terms that are defined and explained in detail in the Jury Instructions. Please refer to the Jury Instructions if you are unsure about the meaning or usage of any legal term that appears in the questions below.

We, the jury, unanimously agree to the answers to the following questions and return them under the instructions of this court as our verdict in this case.

**QUESTION 1: Infringement**

Did Express Mobile prove by a preponderance of the evidence that GoDaddy infringes the identified claim of the Asserted Patents?

Please check "Yes" or "No" for each claim. "Yes" is in favor of Express Mobile and "No" is in favor of GoDaddy.

<b>'755 Patent</b>	Claim 1: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 3: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 12: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 16: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 22: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy)	
<b>'287 Patent</b>	Claim 1: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 13: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy)	
<b>'044 Patent</b>	Claim 1: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 11: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 13: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 17: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy) Claim 19: Yes: ____ (for Express Mobile) No: <u>✓</u> (for GoDaddy)	

**Please Proceed to Question 2.**

**QUESTION 2: Willfulness**

Did Express Mobile prove by a preponderance of the evidence that GoDaddy willfully infringes the identified claim of the Asserted Patents?

<b>'755 Patent</b>	Claim 1:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 3:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 12:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 16:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 22:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
<b>'287 Patent</b>	Claim 1:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 13:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
<b>'044 Patent</b>	Claim 1:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 11:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 13:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 17:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)
	Claim 19:	Yes: ____ (for Express Mobile)	No: <input checked="" type="checkbox"/> (for GoDaddy)

**Please Proceed to Question 3.**

**QUESTION 3: Invalidity**

Did GoDaddy prove by clear and convincing evidence that the identified claim is invalid?

Please check "Yes" or "No" for each claim. "Yes" is in favor of GoDaddy and "No" is in favor of Express Mobile.

<b>'755 Patent</b>	Claim 1:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 3:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 12:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 16:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 22:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
<b>'287 Patent</b>	Claim 1:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 13:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
<b>'044 Patent</b>	Claim 1:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 11:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 13:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 17:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)
	Claim 19:	Yes: ____ (for GoDaddy)	No: <input checked="" type="checkbox"/> (for Express Mobile)

If you checked "Yes" for any claims in Question 1 and checked "No" for any such claims in in Question 3, i.e., you determined that at least one claim is infringed and not invalid, proceed to Question 4.

Otherwise, skip and DO NOT answer Question 4, and instead please proceed directly to the Final Page of the Jury Verdict and sign and date that page.

**QUESTION 4: Damages**

What sum of money in the form of a reasonable royalty do you find that Express Mobile proved by a preponderance of the evidence would fairly and reasonably compensate Express Mobile for GoDaddy's past infringement of Express Mobile's Asserted Patents?

\$ \_\_\_\_\_

Is that sum a (check one):

\_\_\_\_\_ fully paid-up lump sum; or

\_\_\_\_\_ running royalty through trial (February 27, 2023)?

**Please proceed to the Final Page of the Verdict Form and sign and date that page.**

**Final Page of the Jury Verdict**

You have now reached the end of the verdict form and should review it to ensure it accurately reflects your unanimous determinations. The foreperson should then sign and date the verdict form in the spaces below and notify the Court Security Officer that you have reached a verdict.

The foreperson should retain possession of the verdict form and bring it when the jury is brought back into the courtroom.

A black rectangular box redacting the signature of the foreperson.

Signature

3/3/23

Date



**IN THE UNITED STATES DISTRICT COURT  
FOR THE DISTRICT OF DELAWARE**

---

EXPRESS MOBILE, INC.,

Plaintiff,

v.

GODADDY.COM, LLC,

Defendant.

---

)  
)  
)  
)  
) Civil Action No.1:19-cv-01937-MFK  
)  
)  
)  
)  
)

**JUDGMENT AFTER VERDICT**

On August 8, 2022, the Court granted Defendant GoDaddy.com, LLC's ("GoDaddy's") motion for summary judgment of non-infringement of the asserted claims of U.S. Patent No. 6,546,397, and U.S. Patent No. 7,594,168, and hereby enters judgment that GoDaddy does not infringe claims 1, 2, 3, 11 and 37 of U.S. Patent No. 6,546,397 or claims 1, 2, and 3 of U.S. Patent No. 7,594,168.

On February 27, 2023, the Court held a jury trial, and the jury returned its verdict on March 3, 2023. Pursuant to Federal Rule of Civil Procedure 58(b), the Court hereby enters judgment that:

1. GoDaddy does not infringe claims 1, 3, 12, 16, and 22 of U.S. Patent No. 9,063,755; claims 1 and 13 of U.S. Patent No. 9,471,287; and claims 1, 11, 13, 17, and 19 of U.S. Patent No. 9,928,044; and

2. Claims 1, 3, 12, 16, and 22 of U.S. Patent No. 9,063,755; claims 1 and 13 of U.S. Patent No. 9,471,287; and claims 1, 11, 13, 17, and 19 of U.S. Patent No. 9,928,044 are not invalid.

The parties agree to the form of judgment set forth above and reserve all rights with respect to costs, post-trial motions, and appeal.

Dated: March 6, 2023

/s/ Timothy Devlin

Timothy Devlin (No. 4241)  
tdevlin@devlinlawfirm.com  
DEVLIN LAW FIRM LLC  
1526 Gilpin Avenue  
Wilmington, Delaware 19806  
Telephone: (302) 449-9010  
Facsimile: (302) 353-4251

OF COUNSEL:

James R. Nuttall (*pro hac vice*)  
John L. Abramic (*pro hac vice*)  
Michael Dockterman (*pro hac vice*)  
Katherine H. Tellez (*pro hac vice*)  
Robert F. Kappers (*pro hac vice*)  
Tron Fu (*pro hac vice*)  
Daniel F. Gelwicks (*pro hac vice*)  
jnuttall@steptoe.com  
jabramic@steptoe.com  
mdockterman@steptoe.com  
ktellez@steptoe.com  
rkappers@steptoe.com  
tfu@steptoe.com  
dgelwicks@steptoe.com

STEPTOE & JOHNSON LLP  
227 West Monroe, Suite 4700  
Chicago, IL 60606  
(312) 577-1300

Christopher A. Suarez (*pro hac vice*)  
csuarez@steptoe.com  
STEPTOE & JOHNSON LLP  
1330 Connecticut Avenue, NW  
Washington, DC 20036  
(202) 429-8131

*Attorneys for Plaintiff Express Mobile, Inc.*

/s/ Brian S.S. Auerbach

Beth Moskow-Schnoll (No. 2900)  
Brittany Giusini (No. 6034)  
Brian S.S. Auerbach (No. 6532)  
BALLARD SPAHR LLP  
919 N. Market Street, 11th Floor  
Wilmington, DE 19801-3034  
Telephone: (302) 252-4465  
moskowb@ballardspahr.com  
giusinib@ballardspahr.com  
auerbachb@ballardspahr.com

OF COUNSEL:

Brian W. LaCorte (*pro hac vice*)  
Jonathon A. Talcott (*pro hac vice*)  
Andrew H. Hensley (*pro hac vice*)  
1 East Washington Street, Suite 2300  
Phoenix, AZ 85004-2555  
Telephone: (602) 798-5400  
lacorteb@ballardspahr.com  
talcottj@ballardspahr.com  
hensleya@ballardspahr.com

Lawrence Nodine (*pro hac vice*)  
Kyle Ceuninck (*pro hac vice*)  
999 Peachtree Street, NE, Suite 1600  
Atlanta, GA 30309-3915  
(678) 420-9362  
nodinel@ballardspahr.com  
ceuninckk@ballardspahr.com

Caryn Borg-Breen (*pro hac vice*)  
1735 Market Street, 51<sup>st</sup> Floor  
Philadelphia, PA 19103  
(215) 864-8841  
borgbreenc@ballardspahr.com

*Attorneys for Defendant GoDaddy.com, LLC*

IT IS SO ORDERED this 6th day of March, 2023:

  
The Honorable Judge Matthew F. Kennelly

EXPRESS MOBILE, INC., )  
)  
Plaintiff, )  
)  
vs. ) Civil Action No. 1:19-cv-01937-MFK  
)  
GODADDY.COM, LLC, )  
)  
Defendant. )

## Appx120

has also moved for a new trial under Federal Rule of Civil Procedure 59(a). For the reasons set forth below, the Court denies both motions.

### **Background**

The Court assumes familiarity with this case's factual and procedural background, which the Court has discussed in prior written opinions. The following background is relevant to the post-trial motions and largely is taken from the Court's summary judgment decision.

#### **A. The asserted patents**

The asserted patents relate to two concepts that allow web designers to integrate third-party web services into their websites. The patents disclose an authoring tool that generates two sets of code, a device-independent application and a device-specific player. Second, the patents disclose an authoring tool that allows for integration of web services into the application and player.

#### **B. The accused products**

Two GoDaddy products are at issue in this case. WSB is a browser-based website creator, designed to allow a user with little or no website design experience to quickly create and publish a website. MWP, in contrast, targets more advanced users. It comprises a set of custom functionalities that GoDaddy has added to WordPress, an open-source (i.e. free) platform for creating websites. Both WSB and MWP allow for incorporation of features into a user's website, such as YouTube videos or embedded Twitter feeds. GoDaddy provides several free themes for use in designing a website, but users can also buy third-party themes for use with the MWP platform.

#### **C. Claim construction**

Two orders have previously been issued in this case construing certain disputed claim terms. *Express Mobile, Inc. v. GoDaddy.com, LLC*, No. 19-1937-RGA, 2021 WL 2209868 (D. Del. Jun. 1, 2021), and *Express Mobile, Inc. v. GoDaddy.com, LLC*, No. 19-1937-MFK, Mem. Op. and Order (dkt. no. 261) (D. Del. Aug. 8, 2022). The claim terms relevant to the issues decided at trial were construed as follows:

registry	"a database that is used for computing functionality"
player	"device-specific code which contains instructions of a device and which is separate from the Application"
device-dependent code	code that is specific to the operating system, programming language, or platform of a device

#### **D. Evidence at trial**

As relevant to the contentions Express Mobile has made in its motions, the following witnesses testified during the trial, either live or via deposition:

- Dr. Kevin Almeroth, Express Mobile's infringement expert
- Peter Kent, GoDaddy's infringement expert
- Steven Rempell, CEO of Express Mobile
- Ken Brown, Inventor of the asserted patents
- Franklin Jarrett, GoDaddy engineer
- Aaron Silvas, GoDaddy engineer

On a high level, GoDaddy argued that Express Mobile made a bad bet when it filed its patent applications back in 2008, as the patents are directed to device-specific players that have since been rendered obsolete. GoDaddy argued that, by contrast, its WSB and MWP products operate not via players but on browsers and that they use programming language such as JavaScript that is completely device-independent, which GoDaddy argued is necessary in the ever-evolving technological space of smartphones.

The asserted patents recite nineteen claim elements. Kent testified, and GoDaddy argued based on his testimony, that neither WSB nor MWP infringe the asserted patents because they do not include three of the nineteen required claim elements: (1) "player" code that receives web service output; (2) "registry"; or (3) "symbolic names."

Regarding the player limitation, Dr. Almeroth testified, and Express Mobile argued based on his testimony, that WSB and MWP include device-dependent JavaScript player code that branches and accounts for different browser platforms. Because the player is also required to "receive the output symbolic name and corresponding one or more output values and provide[] instructions for a display of the device to present an output value," JTX-001 at claim 1, and because GoDaddy's products are browser-based, Dr. Almeroth testified that WSP and MWP infringe because their JavaScript players contain browser detection code that work together with other code to receive inputs and display outputs.

GoDaddy, for its part, argued that JavaScript—the primary programming language used in WSP and MWP—can work on all browsers and is thus not device-dependent-code. This argument was based in part on Kent's testimony that a browser cannot constitute a platform under the Court's construction of the "player" claim term. GoDaddy also argued that Dr. Almeroth confirmed on cross examination that the only specific code segments in each of the files that Express Mobile contends received the web service outputs were not, in fact, device-dependent code.

Regarding symbolic names, Dr. Almeroth testified, and Express Mobile argued based on his testimony, that WSB and MWP use a registry that stores symbolic names,

and that those symbolic names are incorporated into HTML code that is sent from GoDaddy's servers to its customers' web browsers. It further argued that the symbolic names stored in the registry were used for evoking web components.

By contrast, Kent testified, and GoDaddy argued based on his testimony, that to be a symbolic name, the name must evoke one or more web components and must be stored in a registry—a claim term that the Court construed as "a database used for computing information." Thus, GoDaddy argued, because the existence of a registry is required for the symbolic name claim element to be infringed (and vice versa)—and because the accused symbolic name does not evoke one or more web components or is not stored in a database—then there is no infringement. Kent explained that what Dr. Almeroth pointed to as symbolic names were actually something called "div IDs," which are randomly generated and thus cannot be stored in a registry. Kent also explained that, to the extent symbolic names or registry exist in WSB or MWP at all, they are kept and stored by the external web services intended to be accessed—like YouTube—and not by GoDaddy.

## **Discussion**

### **A. Applicable law**

For issues that are not patent-law-specific, the Federal Circuit defers to the law of the regional circuit in which the district court sits—here, the United States Court of Appeals for the Third Circuit. *Harris Corp. v. Ericsson Inc.*, 417 F.3d 1241, 1248 (Fed. Cir. 2005); *see also, In re Cambridge Biotech Corp.*, 186 F.3d 1356, 1367 (Fed. Cir. 1999) ("we apply our own law with respect to issues of substantive patent law and certain procedural issues pertaining to patent law, but as to nonpatent issues, we apply

the law of the circuit in which the district court sits."). The Federal Circuit "review[s] the denial or grant of JMOL under regional circuit law." *ActiveVideo Networks, Inc. v. Verizon Commc'ns, Inc.*, 694 F.3d 1312, 1319 (Fed. Cir. 2012).

#### **B. Standard for JMOL**

Express Mobile has moved for judgment as a matter of law (JMOL) on all of its infringement claims, or in the alternative for a new trial. Under Federal Rule of Civil Procedure 50(b), JMOL is proper only if "viewing the evidence in the light most favorable to the nonmovant and giving it the advantage of every fair and reasonable inference, there is insufficient evidence from which a jury reasonably could find" for the nonmovant. *Lightning Lube, Inc. v. Witco Corp.*, 4 F.3d 1153, 1166 (3d Cir. 1993) (citing *Wittekamp v. Gulf & Western Inc.*, 991 F.2d 1137, 1141 (3d Cir. 1993)). "The question is not whether there is literally no evidence supporting the party against whom the motion is directed but whether there is evidence upon which the jury could properly find a verdict for that party." *Id.* (quoting *Patzig v. O'Neil*, 577 F.2d 841, 846 (3d Cir. 1978)).

To grant JMOL on infringement in favor of Express Mobile, the party with the burden of proof, the Court "must be able to say not only that there is sufficient evidence to support the finding [sought by Express Mobile] . . . but additionally that there is *insufficient evidence for permitting any different finding.*" *Fireman's Fund Ins. Co. v. Videofreeze Corp.*, 540 F.2d 1171, 1177 (3d Cir. 1976) (emphasis added); *see also*, *Smollett v. Skayting Dev. Corp.*, 793 F.2d 547, 548 (3d Cir. 1986) (motion for JMOL must be denied when "the record contains the minimum quantum of evidence from which a jury might reasonably afford relief").



In assessing Express Mobile's motion, the Court must also give GoDaddy, the non-movant, "the benefit of all logical inferences that could be drawn from the evidence presented" and "resolve all conflicts in the evidence in [GoDaddy's] favor and, in general, view the record in the light most favorable to [GoDaddy]." *Williamson v. Consol. Rail Corp.*, 926 F.2d 1344, 1348 (3d Cir. 1991).

The Third Circuit has held that motions for judgment as a matter of law under Rule 50(b) should be granted only sparingly. *Marra v. Phila. Hous. Auth.*, 497 F.3d 286, 300 (3d Cir. 2007) (citation omitted). This is particularly true when, as just discussed, the movant bears the burden of proof. *Fireman's Fund Ins. Co.*, 540 F.2d at 1177 (entry of JMOL after a jury verdict "is rare[ ] [and] reserved for extreme circumstances.").

#### **C. Standard for a new trial**

The Court has discretion to grant a new trial under Rule 59(a) "for any of the reasons for which new trials have heretofore been granted in actions at law in the courts of the United States." Fed. R. Civ. P. 59(a). Some reasons for granting a new trial are: (1) the jury's verdict is against the clear weight of the evidence and a new trial is necessary to prevent a miscarriage of justice; (2) improper conduct by an attorney or the Court unfairly influenced the verdict; or (3) the jury's verdict was facially inconsistent. *See Ateliers de la Haute-Garonne v. Broetje Automation-USA Inc.*, 85 F. Supp. 3d 768, 775 (D. Del. 2015).

#### **D. Summary of Express Mobile's motion**

Express Mobile raises two grounds for JMOL or, in the alternative, a new trial.<sup>2</sup>

---

<sup>2</sup> Express Mobile also appears to raise an amorphous third basis, namely that GoDaddy confused the jury with irrelevant "strawmen," Mot. for J. as a Matter of Law at 16, such

First, it contends that GoDaddy misled the jury by consistently misstating and contradicting the Court's claim constructions for player, device-dependent code, and registry. Express Mobile contends that, under the correct claim constructions adopted by the Court, no reasonable jury could have found that GoDaddy did not infringe the asserted patents. Second, Express Mobile contends that the jury's verdict was unfairly influenced by improper testimony from two witnesses that GoDaddy called as fact witnesses but who Express Mobile contends provided undisclosed expert testimony. The Court will address each of these arguments in turn.

**E. Claim construction-related issues**

"[T]he claims of a patent define the invention to which the patentee is entitled the right to exclude." *Innova/Pure Water, Inc. v. Safari Water Filtration Sys., Inc.*, 381 F.3d 1111, 1115 (Fed. Cir. 2004). "Literal infringement of a claim exists when every limitation recited in the claim is found in the accused device." *Kahn v. Gen. Motors Corp.*, 135 F.3d 1472, 1477 (Fed. Cir. 1998) (emphasis added).

Express Mobile contends that it presented to the jury uncontradicted evidence that all of the asserted claims' limitations—including the three that GoDaddy argued were missing from its products—were met by WSB and MWP. Specifically, it contends that Dr. Almeroth "identif[ied] exemplary portions of source code and related evidence for each and every limitation of all Asserted Claims for both WSB and MWP, with a particular focus on contact form and add to cart web service implementations of the

---

as: (1) its overuse of the YouTube web service example to illustrate some of its arguments regarding the symbolic names limitation; and (2) its arguments regarding div IDs. But it offers no authority for the proposition that "red herring"-type arguments—if that characterization is even accurate—would entitle it to either JMOL or a new trial. *Id.*

respective products." Mot. for J. as a Matter of Law at 4. Thus, it contends that there is no evidentiary basis in the record on which the jury could have found non-infringement for any of the disputed claim elements.

As for GoDaddy's evidence showing non-infringement, Express Mobile contends that it was premised upon constructions of claim terms that contradicted the Court's claim construction rulings. It is settled law that "[n]o party may contradict the court's construction to a jury." *Exergen Corp. v. Wal-Mart Stores, Inc.*, 575 F.3d 1312, 1321 (Fed. Cir. 2009).

Regarding the player limitation, Express Mobile contends that GoDaddy argued—contrary to the Court's claim construction of device-dependent code as "code that is specific to the operating system, programming language, or platform of a device"—that its JavaScript player was not a device-dependent code because device-dependent code is limited to code specific to the *operating system* of a device. This, Express Mobile contends, obviates the "programming language, or platform" language in the Court's construction. This is significant, Express Mobile says, because it argued that the accused *browser* constitutes a "platform" under the Court's construction, which means the browser's code is device-dependent code and thus a player.

Next is the registry limitation, construed by the Court as meaning "a database that is used for computing functionality." Express Mobile contends that GoDaddy improperly narrowed the term's construction to a *structured* database. Express Mobile contends that a "database," according to the Court's construction of the registry term, can include a file and need not be structured.

There are multiple problems with Express Mobile's arguments. First, the Court is

not persuaded that GoDaddy presented a modified or narrowed version of the Court's claim construction to the jury. GoDaddy consistently included programming language or platforms in its arguments at trial regarding the "device-dependent code" term. See, e.g., Trial Tr. at 1024-26 (explaining during closing arguments how device specific code encompasses code specific to certain operating systems, programming languages, or platforms). And contrary to Express Mobile's contention, GoDaddy expert witness Kent also testified that, consistent with the Court's construction, device-dependent code could include dependency on the operating system, programming language, or platform, not merely on an operating system. *Id.* at 618:12-619:23, 676:16-23, 697:23-698:10, and 699:24-700:8. On the "registry" term, contrary to Express Mobile's view, the Court's construction does not *require* it to include a simple, unstructured database. In addition, as GoDaddy points out, the slides it showed to the jury and used during Kent's testimony could not have more clearly spelled out each of the Court's claim constructions that Kent was using to render his opinions on infringement of those claims.

Second, Express Mobile is essentially raising a claim construction argument regarding the meaning of the terms "platform" and "database" under the guise of a challenge to the sufficiency of the evidence of noninfringement. That is, Express Mobile's argument invites the Court to hold that a browser may constitute a platform, and vice versa, and that a simple file may constitute a database. But the Court did not construe the terms platform or database. "In the absence of such a construction . . . the jury was free to rely on the plain and ordinary meaning of the term[s] [platform and database] and conclude"—consistent with Kent's testimony—that a browser is not a

platform under the Court's construction of device-dependent code (and therefore, player), and that a file is not a database under the Court's construction of registry. *ePlus, Inc. v. Lawson Software, Inc.*, 700 F.3d 509, 520 (Fed. Cir. 2012). Moreover, "[a] sound claim construction need not always purge every shred of ambiguity, including potential ambiguity arising from the words a court uses to construe a claim term." *Rembrandt Wireless Techs., LP v. Samsung Elecs. Co.*, 853 F.3d 1370, 1378–79 (Fed. Cir. 2017) (internal quotation marks omitted). "Such an endeavor could proceed ad infinitum." *Eon Corp. IP Holdings v. Silver Spring Networks*, 815 F.3d 1314, 1318 (Fed. Cir. 2016).

Third, even assuming that Express Mobile is correct that GoDaddy misled the jury about the proper claim constructions, that would not automatically entitle Express Mobile to JMOL. "Where an [non]infringement verdict relies on incorrect construction of the disputed claim terms, this court may grant JMOL or order a new trial to correct the error, *depending on the degree of difference between the incorrect construction and the correct construction.*" *Finisar Corp. v. DirecTV Grp., Inc.*, 523 F.3d 1323, 1333 (Fed. Cir. 2008). Express Mobile has not persuaded the Court that the degree of difference between the allegedly incorrect constructions used by GoDaddy/Kent and the Court's constructions—if there is even a difference—is significant enough to entitle it to JMOL. To hold otherwise would "constitute a usurpation of the jury's province as factfinder." *Garrison v. Mollers N. Am., Inc.*, 820 F. Supp. 814, 819 (D. Del. 1993).

Express Mobile's other argument on this point is that "[u]nder the correct claim constructions, Express Mobile presented undisputed evidence that WSB and MWP" infringe the asserted patents, and that "no reasonable jury could have concluded

otherwise." Mot. for J. as a Matter of Law at 2. But the evidence presented by Express Mobile was disputed, and "when there is conflicting testimony at trial, and the evidence overall does not make only one finding on the point reasonable, the jury is permitted to make credibility determinations and believe the witness it considers more trustworthy." *MobileMedia Ideas LLC v. Apple Inc.*, 780 F.3d 1159, 1168 (Fed. Cir. 2015). When evaluating a motion for JMOL, "the court may not weigh the evidence, determine the credibility of witnesses, or substitute its version of the facts for the jury's version." *Lightning Lube, Inc.*, 4 F.3d at 1166.

Express Mobile cites *Moba, B.V. v. Diamond Automation, Inc.*, 325 F.3d 1306, (Fed. Cir. 2003), and contends that, in that case, the Federal Circuit "revers[ed] denial of JMOL where 'the district court allowed the jury to add an additional limitation to the district court's construction' and this error altered the verdict because there was 'no alternative basis upon which a reasonable jury could find' noninfringement." Mot. for J. as a Matter of Law at 11 (quoting *Moba, B.V.*, 325 F.3d at 1322). But even if GoDaddy did add additional limitations to the Court's construction—which, again, the Court is not persuaded it did—in this case there *were* alternative bases upon which a reasonable jury could find noninfringement. Again, GoDaddy argued that its products were missing *three* of the required claim elements taught by the patents, and a finding of noninfringement on any *one* of those claim elements would have entitled GoDaddy to the same verdict. It is not enough for Express Mobile to argue the jury *should* have found in its favor, it must establish "there is insufficient evidence for permitting *any different finding*" other than that of infringement. *Fireman's Fund Ins. Co.*, 540 F.2d at 1171 (emphasis added). Express Mobile has not made that showing.

In sum, the Court presumes, as it must at this stage, that the jury credited Kent's testimony over Dr. Almeroth's in finding that the accused instrumentalities did not infringe the asserted claims as construed by the Court—specifically that the WSB and MWP platforms did not include: (1) "player" code, (2) "symbolic names," or (3) a "registry." The trial record demonstrates that the jury received sufficient evidence from which it could reasonably find that GoDaddy's products do not infringe. The Court therefore denies Express Mobile's renewed motion for JMOL on this basis.

The next question is whether the cited conduct entitles Express Mobile to a new trial. The Court is not persuaded that a new trial is warranted. As Express Mobile argues, "[a] court may grant a new trial when 'the jury's verdict is against the clear weight of the evidence, and a new trial must be granted to prevent a miscarriage of justice.'" Mot. for J. as a Matter of Law at 3 (quoting *Genzyme Corp. v. Atrium Med. Corp.*, 315 F. Supp. 2d 552, 562 (D. Del. 2004)). Express Mobile also contends that "[a] court may . . . grant a new trial 'when improper conduct by an attorney or the court unfairly influenced the verdict.'" *Id.* (quoting *Zarow-Smith v. New Jersey Transit Rail Operations, Inc.*, 953 F. Supp. 581, 584–85 (D.N.J. 1997)). Neither situation is present here. As just discussed, the clear weight of the evidence did not lean in favor of a verdict of infringement. Nor were the arguments made by GoDaddy, when considered "as a whole," "so constantly and effectively addressed to the prejudices of the jury that [the Court] must order a new trial." *Draper v. Airco, Inc.*, 580 F.2d 91, 97 (3d Cir. 1978).

**F. Undisclosed expert witness testimony**

Express Mobile next contends that it is entitled to JMOL or, in the alternative, a new trial because GoDaddy presented undisclosed expert rebuttal testimony through

two of its fact witnesses that resulted in "jury confusion and error." Mot. for J. as a Matter of Law at 16. The Court sees—and Express Mobile presents—no viable argument for JMOL on this basis, so it will consider Express Mobile's contentions on this point only as support for its motion for a new trial. Express Mobile again contends that "a court may [] grant a new trial 'when improper conduct by an attorney or the court unfairly influenced the verdict,'" *id.* at 3 (quoting *Zarow-Smith*, 953 F. Supp. at 584–85), or based on "substantial errors in admission or rejection of evidence." *Montgomery Ward & Co. v. Duncan*, 311 U.S. 243, 251 (1940).

Under Federal Rule of Civil Procedure 26(a)(2)(A), a party must disclose the identity of a witness who is going to present expert (i.e., non-lay person) testimony.<sup>3</sup> Rule 26(a)(3)(B) requires that disclosure to be made at least thirty days before trial unless the court orders otherwise. A disclosure under Rule 26(a)(2)(A) must be accompanied by a statement of "the subject matter on which the witness is expected to present evidence under Federal Rule of Evidence 702, 703, or 705" and "a summary of the facts and opinions to which the witness is expected to testify." Fed. R. Civ. P. 26(a)(2)(C). A party that fails to make a disclosure required by Rule 26(a) is precluded from relying on that evidence on a motion or at trial unless the failure to disclose was "substantially justified or harmless." Fed. R. Civ. P. 37(c)(1).

During the trial, GoDaddy called as witnesses Aaron Silvas and Franklin Jarrett, two of its engineers who worked on WSP and MWP, respectively. Express Mobile contends that their testimony: (1) ran afoul of Rules 26(a)(2)(A) and 26(a)(2)(C)

---

<sup>3</sup> There are, of course, additional requirements for *retained* witnesses who will present expert testimony, see Fed. R. Civ. P. 26(a)(2)(B), but that is not what is at issue here.



because it was not timely or properly disclosed; and (2) should have been barred because "it is an abuse of discretion to permit a witness to testify as an expert on the issue[] of noninfringement . . . unless that witness is qualified as an expert in the pertinent art . . . [and this] prohibition of unqualified witness testimony extends to the ultimate conclusion[] of infringement . . . as well as to the underlying technical questions." *HVLPO2, LLC v. Oxygen Frog*, 949 F.3d 689, 689 (Fed. Cir. 2020).

Both of these contentions lack merit, as they are based on the mistaken premise that the testimony of Silvas and Jarrett amounted to rebuttal expert testimony. But Express Mobile's argument regarding this testimony fails for an even more basic reason: Express Mobile failed to contemporaneously object to *any* of Silvas and Jarrett's testimony during the trial.

As GoDaddy points out, this post trial motion is not the first time Express Mobile has asserted a challenge to the testimony of Silvas and Jarrett. During the final pretrial conference, the Court heard arguments regarding Express Mobile's motion *in limine* number seven, in which it sought to preclude Jarrett and Silvas from offering expert opinions on GoDaddy's source code. The Court concluded that it could not rule on this in vacuum and that Express Mobile would have to assert any such objections at trial. The Court stated, quite unambiguously, that it was "going to do this the way Judge Andrews did, and it's going to be *based on objections that are made at the trial and everybody basically acts at their peril*. It's just that simple." Pretrial Conf. Tr. at 27:8-24 (emphasis added). The Court went on to say:

So, you know, I'll determine *when I get an objection* whether -- if it concerns the dividing line between fact and opinion testimony, I'll determine based on whatever foundation's been laid, which side of the line it's on. And if I conclude it's on the opinion side of the line, and there's not

a sufficient disclosure under Rule 26(a), then it's probably going to get excluded. And if I conclude it's on the fact side of the line, then it's probably not going to get excluded. So -- but I just think it would be borderline reckless for me to try to define that line right now. I think that's best dealt with by, A, people being careful, and, B, *objections and ruling on the objections*. So that's what I'm going to do.

*Id.* at 27:15–28:2 (emphasis added). The Court therefore denied Express Mobile's motion *in limine* "without prejudice to making objections at trial." *Id.* at 28:3-4 (emphasis added).

On the morning of the third day of trial, before the jury entered the courtroom, Express Mobile again raised its concerns regarding Jarrett and Silvas's testimony. Specifically, counsel for Express Mobile stated that, because Jarrett and Silvas were going to be shown and asked about some of the same exhibits that Dr. Almeroth used to conduct his infringement analysis, it appeared that GoDaddy sought "to implicitly or explicitly, you know, engage in a rebuttal." Trial Tr. at 468:14-16. The Court overruled the objection, concluding that the mere fact that these witnesses would be shown exhibits that had been shown to an expert was not enough to make their testimony expert testimony. The Court further stated, "as it's characterized, I don't think it's expert testimony . . . [and] the jury's basically told they're to evaluate expert testimony and fact testimony the same way." *Id.* at 470:9-13. The Court instructed GoDaddy, however, that it was not to refer to Dr. Almeroth when showing Jarrett or Silvas these exhibits or asking questions about them.

According to Express Mobile, Jarrett and Silvas proceeded to charge ahead past the Court's directive and give what Express Mobile contends was undisclosed expert testimony rebutting that of Dr. Almeroth. But Express Mobile did not make a single contemporaneous objection to their testimony. Again, the Court had clearly instructed

Express Mobile during the pretrial conference that if it believed that Jarrett or Silva were asked during the trial to render expert testimony during the trial, it would be incumbent upon Express Mobile to object at that time. And the Court provided more guidance about what the appropriate boundaries might be on the morning Jarrett and Silvas took the stand, when Express Mobile raised the issue for the second time. But Express Mobile still did not object to either witness's testimony as it was unfolding, as it was required to do.

In fact, the only objection Express Mobile made contemporaneously concerned not the content of Jarrett or Silvas's testimony, but a single exhibit that was not timely disclosed as an exhibit that would be used during Jarrett's testimony.<sup>4</sup> But GoDaddy's failure to include that exhibit—PTX-550D—in Jarrett's exhibit binder is not a violation of the Federal Rules of Civil Procedure that could even conceivably entitle Express Mobile to a new trial. Instead, it was a mere violation of paragraph 47 of the parties' mutual agreement—as contained in the joint pretrial order—regarding advance notice of the exhibits to be used in connection with a direct examination of each witness. That aside, the arguably belated disclosure of this particular exhibit as one that would be used during the witness's testimony could not possibly have unfairly prejudiced Express Mobile. Among other things, the exhibit was on Express Mobile's own list of trial exhibits attached to the proposed pretrial order, thus making it clear that Express Mobile was aware of its possible significance.

Even if Express Mobile had not waived this point by failing to object contemporaneously, the Court is not persuaded that Jarrett or Silvas provided anything

---

<sup>4</sup> GoDaddy has effectively conceded this.

that constitutes expert testimony under Federal Rules of Evidence 702, 703, or 705. A lay witness may testify if he "has particularized knowledge by virtue of [his] experience . . . even if the subject matter is specialized or technical—because the testimony is based upon the layperson's personal knowledge rather than on specialized knowledge within the scope of FRE 702." *Donlin v. Philips Lighting N. Am. Corp.*, 581 F.3d 73, 81 (3d Cir. 2009) (emphasis added). That was the case here. Both witnesses repeatedly testified that they did not review the asserted patents or the Court's claim constructions. And neither voiced any opinion on infringement other than to say, in colloquial terms, that they did not steal or copy someone else's work when they helped develop the code behind WSB (Silvas) and MWP (Jarrett). The quotes that Express Mobile cherry picks from the record are nothing more than Silvas and Jarrett's descriptions, based on their own first-hand knowledge, of how their respective products work or do not work. The Court agrees with GoDaddy that Jarrett and Silvas's mere use of the same terms found in some of the claims at issue does not, by itself, transform their testimony into rebuttal expert testimony on the question of infringement of those claims.

Express Mobile's only argument on this point that even approaches the plausibility threshold involves not Jarrett or Silvas's testimony itself but rather the closing argument by counsel for GoDaddy. Were one to read certain quotes from GoDaddy's closing argument out of context, they conceivably support a belief that GoDaddy characterized Jarrett and Silvas as experts who were called to rebut everything Dr. Almeroth had to say.

Express Mobile's argument is precluded, however, due to its failure to object contemporaneously during GoDaddy's closing argument. It therefore waived this point.

And even if that were not the case, the argument is not supported by a reading of the closing arguments in context and as a whole and thus would not entitle Express Mobile to a new trial. See *Genzyme Corp. v. Atrium Med. Corp.*, 315 F. Supp. 2d 552, 561-62 (D. Del. 2004) (emphasis added) ("[S]tatements made about prior art and infringement in closings that accurately reflected evidence, *which has been admitted without objection*, are not improper."). The comments by counsel for GoDaddy during closing argument accurately reflected the evidence. In particular, there is nothing improper about an argument that a fact witness's testimony discredited the testimony of the opposing party's expert witness. See *Fineman v. Armstrong World Indus., Inc.*, 980 F.2d 171, 207 (3d Cir. 1992) ("not all improper remarks will engender sufficient prejudice to mandate the granting of a new trial" and "a combination of improper remarks are required to persuade [the Court] of prejudicial impact"); see also, *Draper*, 580 F.2d at 97 (in determining whether a new trial must be granted "we do not rely on any of the individual instances or types of impropriety. Rather, we [] assess[] the argument as a whole"). There has been no "miscarriage of justice." *Genzyme Corp.*, 315 F. Supp. 2d at 562.

Express Mobile is not entitled to JMOL or a new trial on this basis.

### Conclusion

For the reasons stated above, the Court denies plaintiff's renewed motion for judgment as a matter of law, or in the alternative, a new trial [dkt. no. 333].

  
MATTHEW F. KENNELLY  
United States District Judge

Date: July 5, 2023



US006546397B1

(12) **United States Patent**  
**Rempell**

(10) **Patent No.:** **US 6,546,397 B1**  
(45) **Date of Patent:** **Apr. 8, 2003**

(54) **BROWSER BASED WEB SITE GENERATION  
TOOL AND RUN TIME ENGINE**

(76) **Inventor:** **Steven H. Rempell**, 38 Washington St.,  
Novato, CA (US) 94947

(\*) **Notice:** Subject to any disclaimer, the term of this  
patent is extended or adjusted under 35  
U.S.C. 154(b) by 0 days.

(21) **Appl. No.:** **09/454,061**

(22) **Filed:** **Dec. 2, 1999**

(51) **Int. Cl.** <sup>7</sup> **G06F 17/00; G06T 13/00**

(52) **U.S. Cl.** **707/102; 707/104.1; 707/501.1;**  
**345/700; 345/473**

(58) **Field of Search** **707/102, 103,**  
**707/104, 501, 513, 517, 530, 104.1, 501.1;**  
**345/333, 967, 326, 339, 348, 352, 700,**  
**473**

(56) **References Cited**

**U.S. PATENT DOCUMENTS**

5,428,731 A \* 6/1995 Powers, III ..... 707/501  
5,842,020 A \* 11/1998 Faustini ..... 395/701  
5,870,767 A \* 2/1999 Kraft, IV ..... 707/501  
6,035,119 A \* 3/2000 Massena et al. .... 717/100  
6,081,263 A \* 6/2000 LeGall et al. .... 345/327  
6,083,276 A \* 7/2000 Davidson et al. .... 717/1

6,148,311 A \* 11/2000 Wishnie et al. .... 707/513  
6,191,786 B1 \* 2/2001 Eyzaguirre et al. .... 345/853

**OTHER PUBLICATIONS**

Piero Fraternali, "Tools and Approaches for Developing  
Data-Intensive Web Applications: A Survey", ACM Sep.  
1999, pp. 226-263.\*

Balusubramanian et al A Large-Scale Hypermedia Applica-  
tion using Document Management and Web Technologies,  
ACM 1997, pp. 134-145.\*

\* cited by examiner

*Primary Examiner*—Safet Metjahic

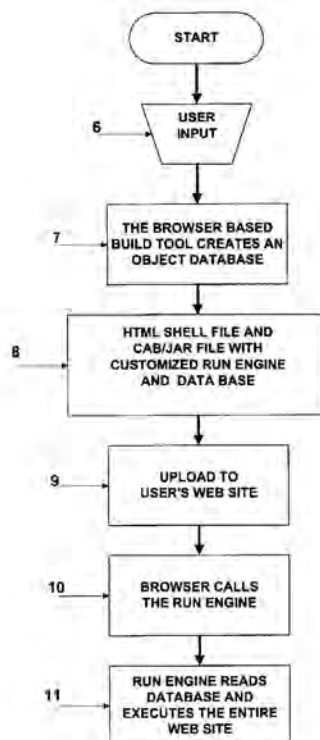
*Assistant Examiner*—Uyen Le

(74) *Attorney, Agent, or Firm*—Coudert Brothers LLP

(57) **ABSTRACT**

A method and apparatus for designing and building a web  
page. The apparatus includes a browser based build engine  
including build tools and a user interface. The build tools are  
operable to construct a single run time file and an associated  
database that describe, and when executed, produce the web  
page. The user interface includes a build frame and a panel.  
The build frame is operable to receive user input and present  
a WYSIWIG representation of the web page. The panel  
includes one or more menus for controlling the form of  
content to be placed on the web page.

**39 Claims, 68 Drawing Sheets**

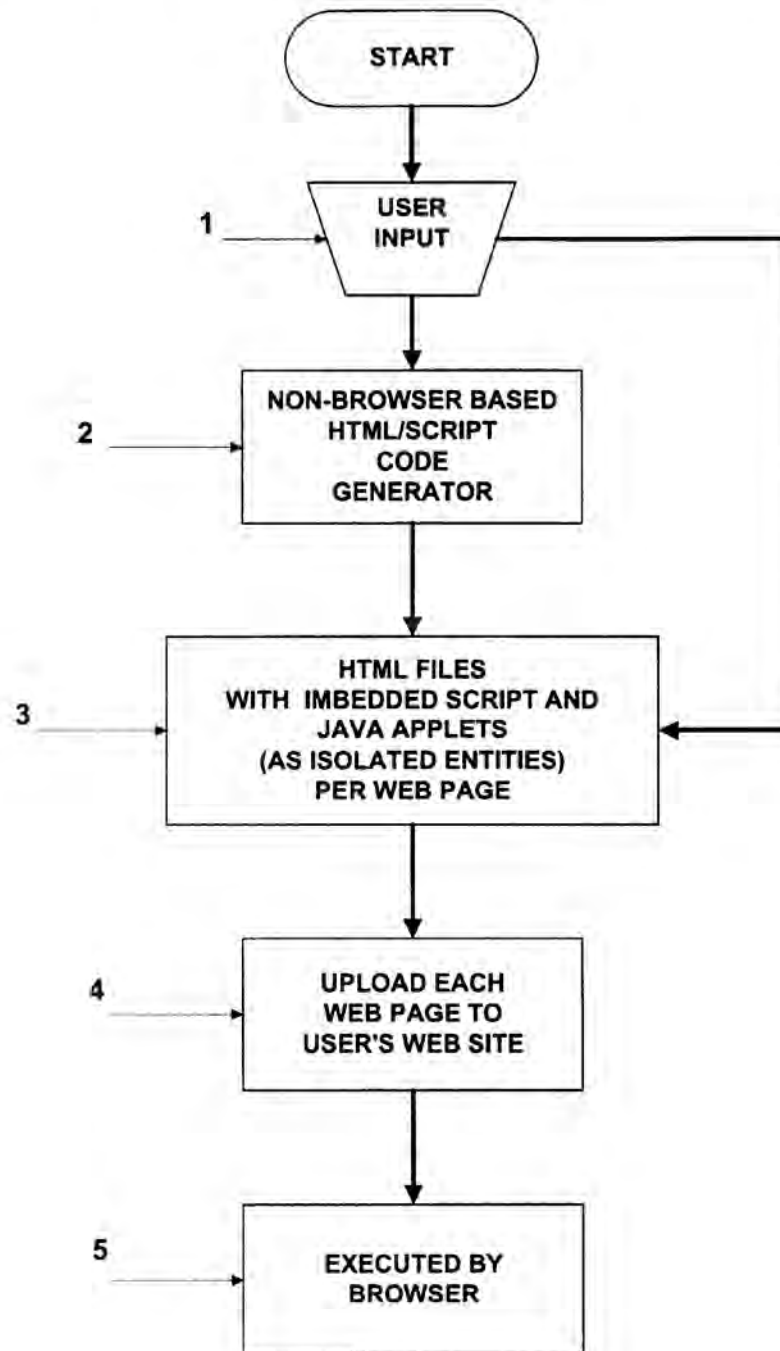


**U.S. Patent**

**Apr. 8, 2003**

**Sheet 1 of 68**

**US 6,546,397 B1**



**Fig. 1**

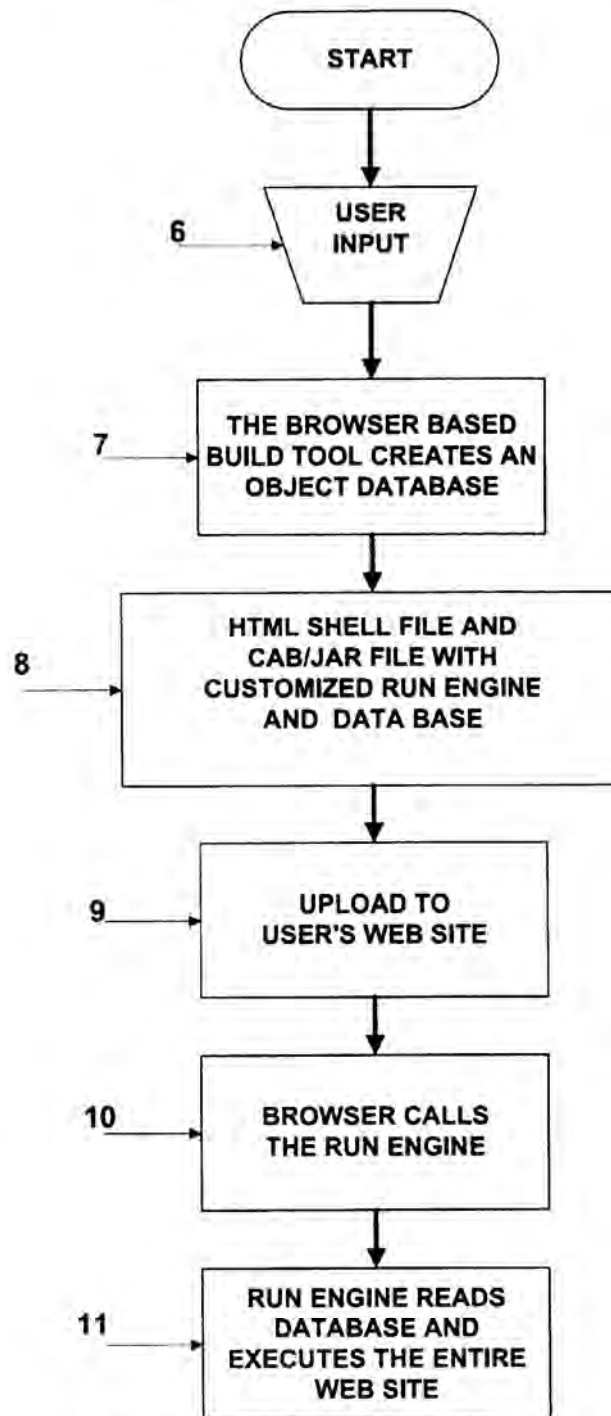
**PRIOR ART**

**U.S. Patent**

**Apr. 8, 2003**

**Sheet 2 of 68**

**US 6,546,397 B1**



**Fig. 2**

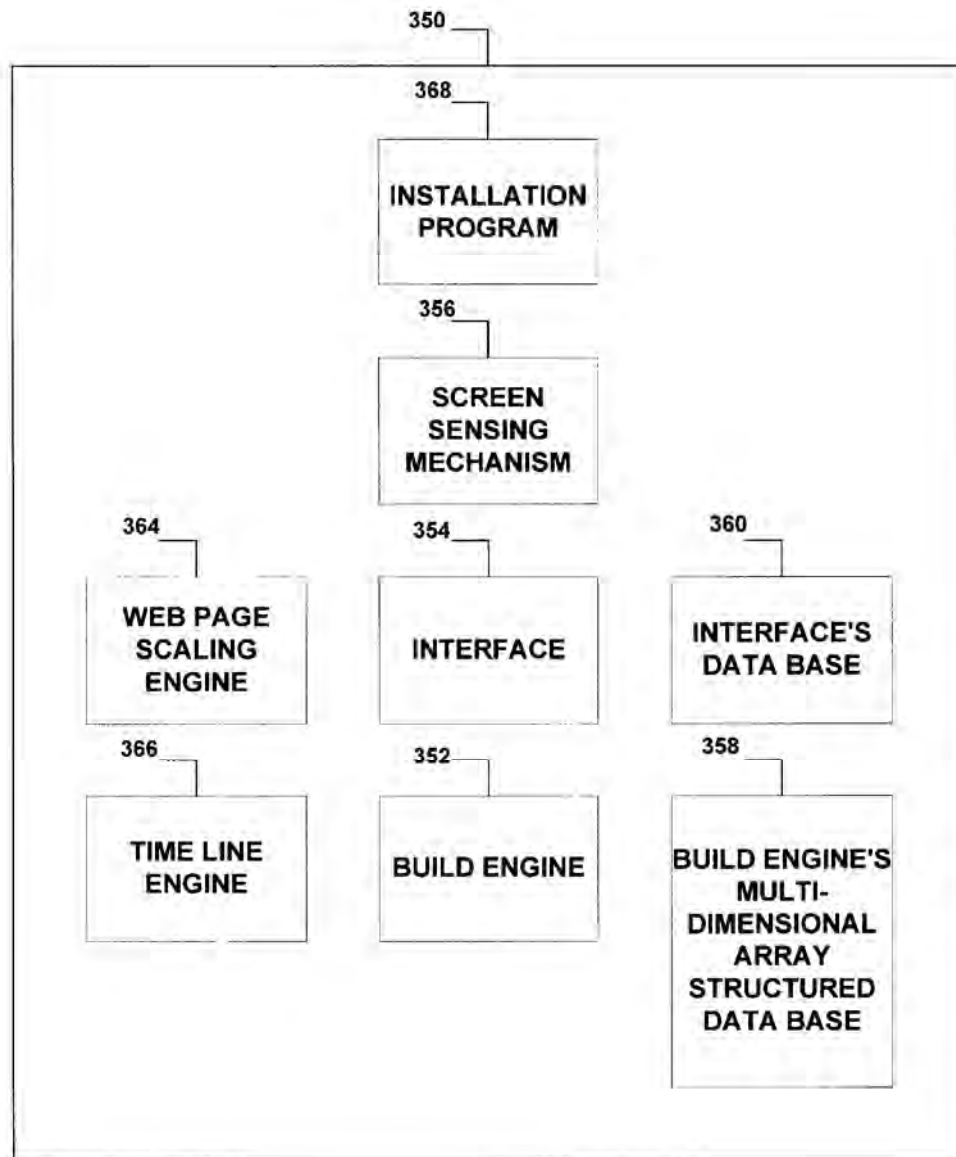


**U.S. Patent**

**Apr. 8, 2003**

**Sheet 3 of 68**

**US 6,546,397 B1**



***Fig. 3a***

**BUILD TOOL COMPONENTS**

U.S. Patent

Apr. 8, 2003

Sheet 4 of 68

US 6,546,397 B1

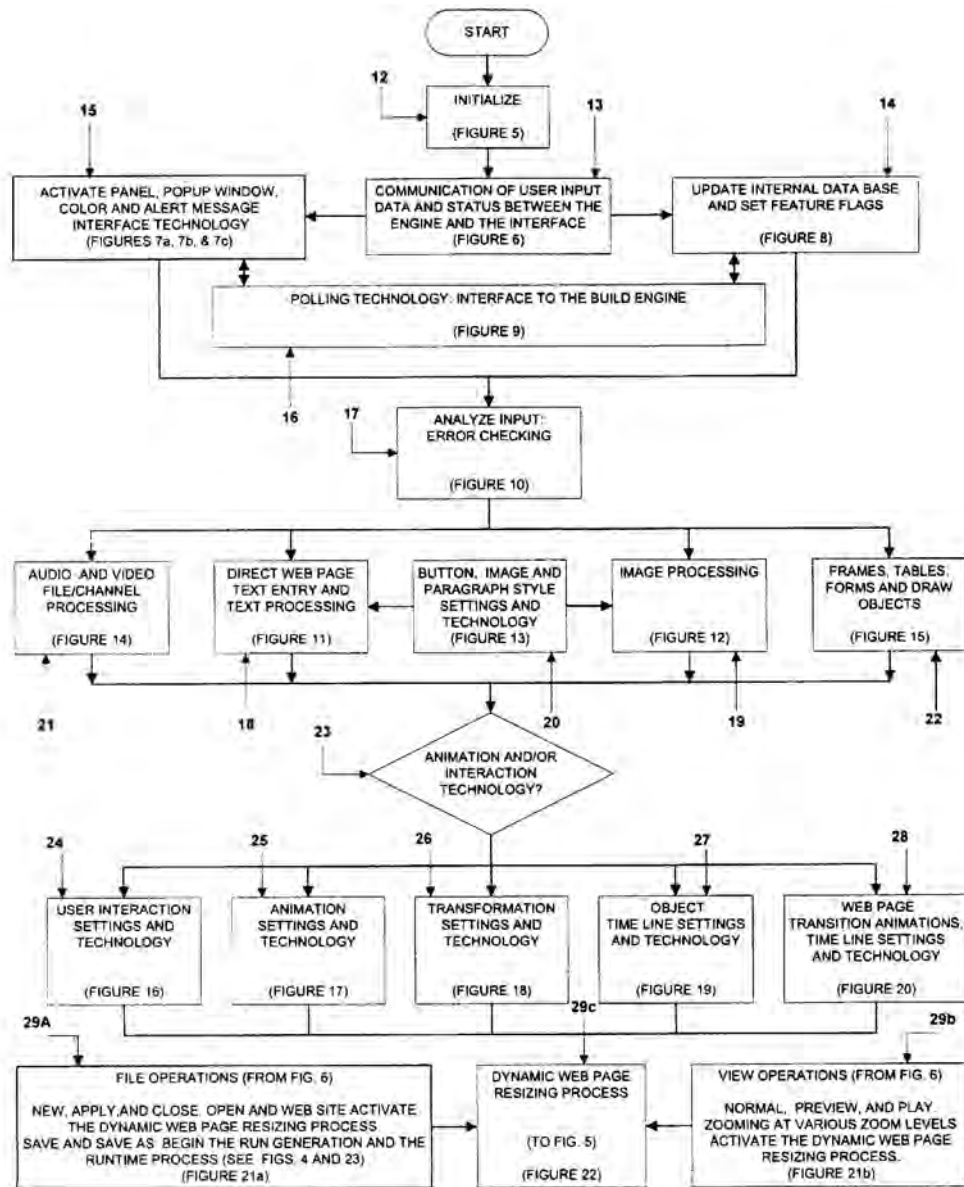


Fig. 3b

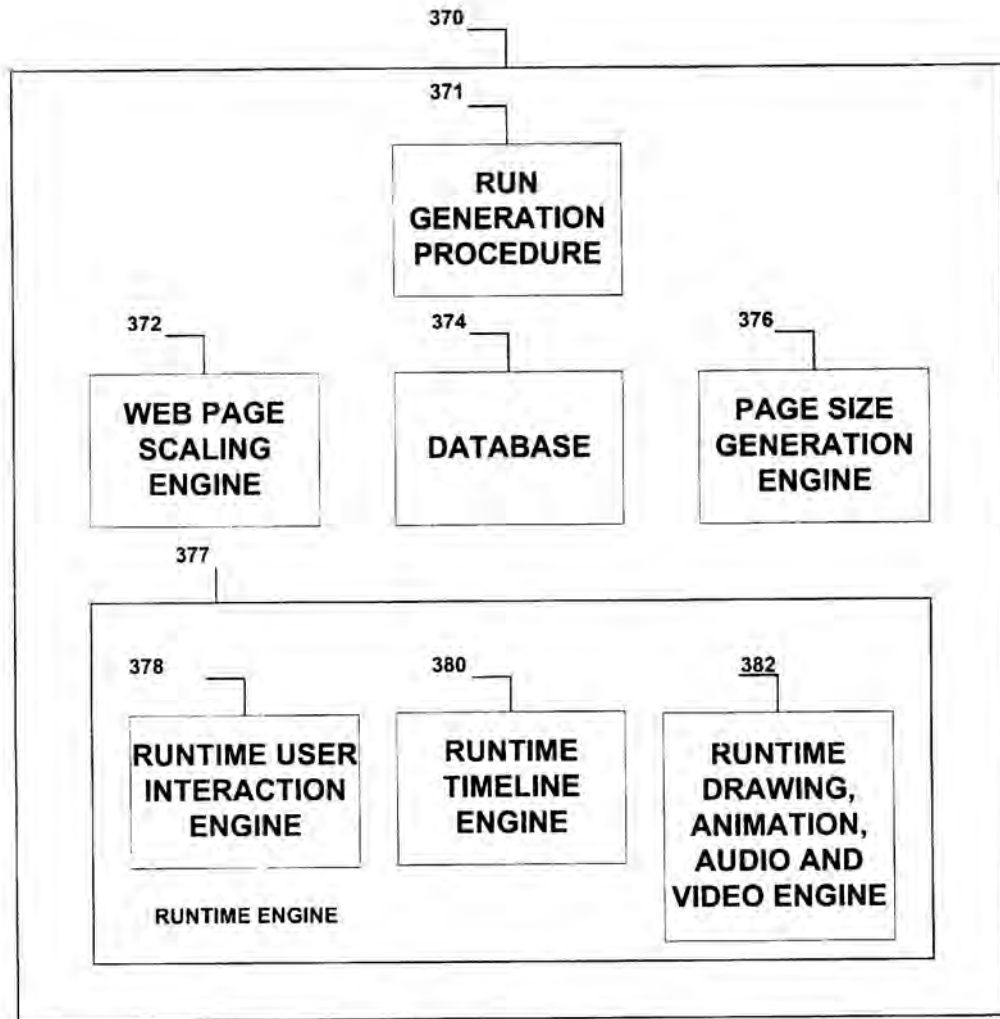
## THE BUILD TOOL &amp; BUILD PROCESS

**U.S. Patent**

**Apr. 8, 2003**

**Sheet 5 of 68**

**US 6,546,397 B1**



***Fig. 4a***

## **RUN GENERATION AND RUNTIME COMPONENTS**

U.S. Patent

Apr. 8, 2003

Sheet 6 of 68

US 6,546,397 B1

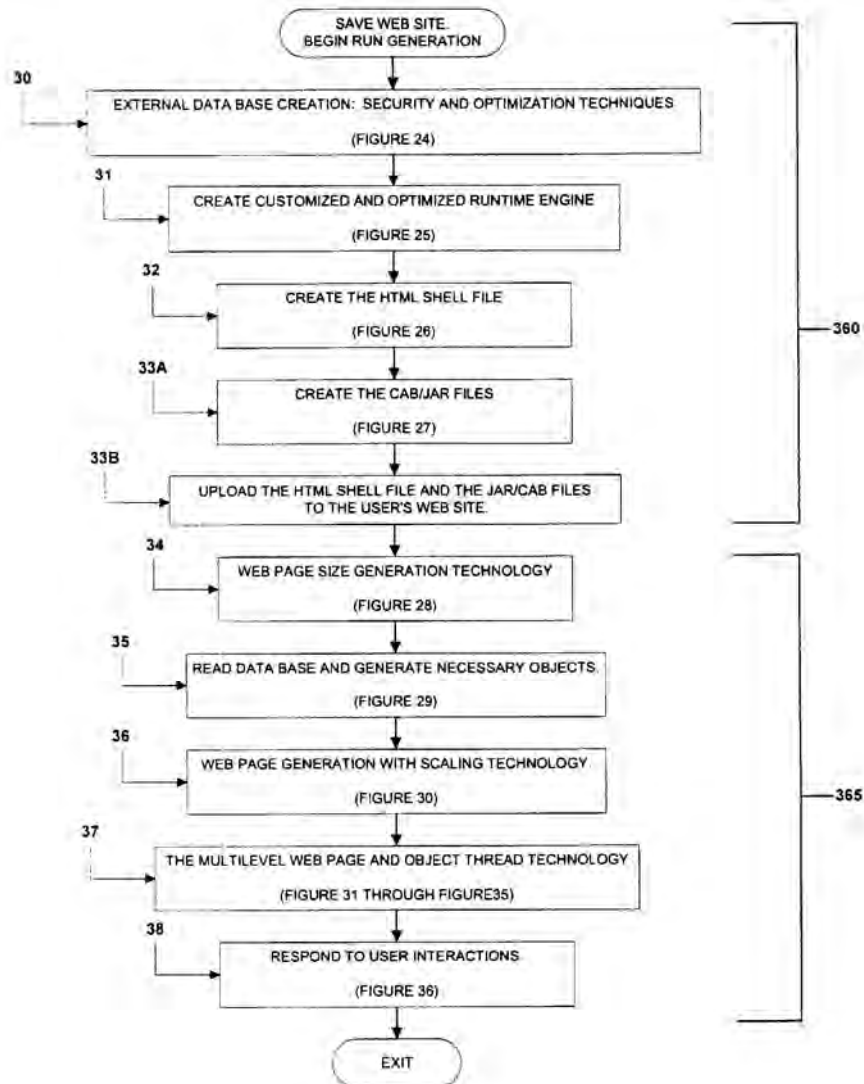


Fig. 4b

**RUN GENERATION & THE RUNTIME PROCESS**

U.S. Patent

Apr. 8, 2003

Sheet 7 of 68

US 6,546,397 B1

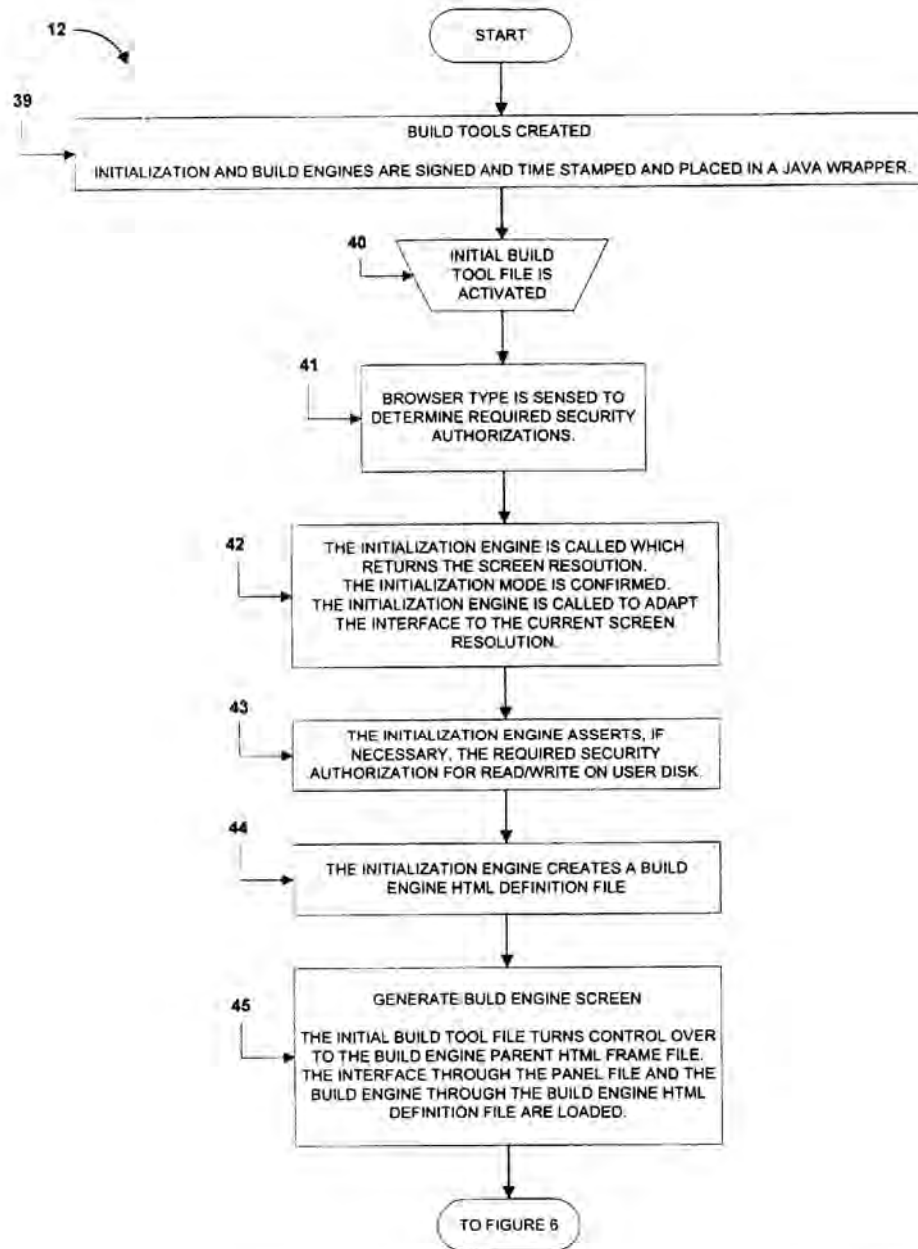


Fig . 5

## INITIALIZATION

U.S. Patent

Apr. 8, 2003

Sheet 8 of 68

US 6,546,397 B1

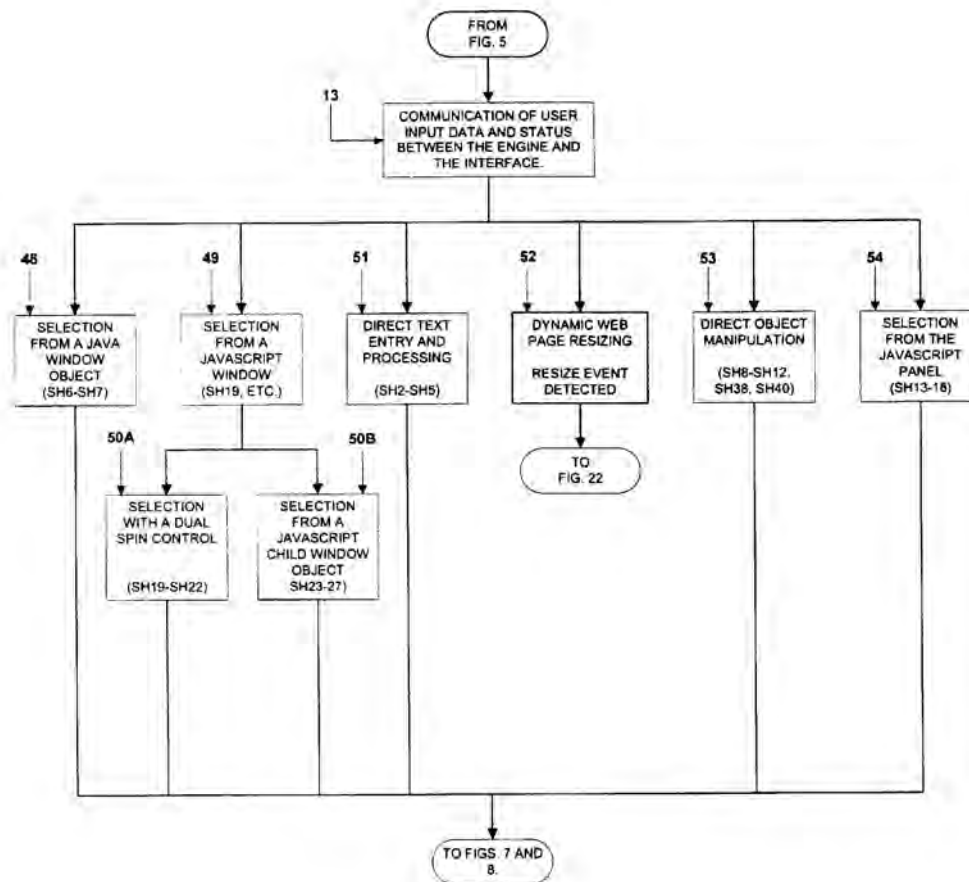


Fig. 6

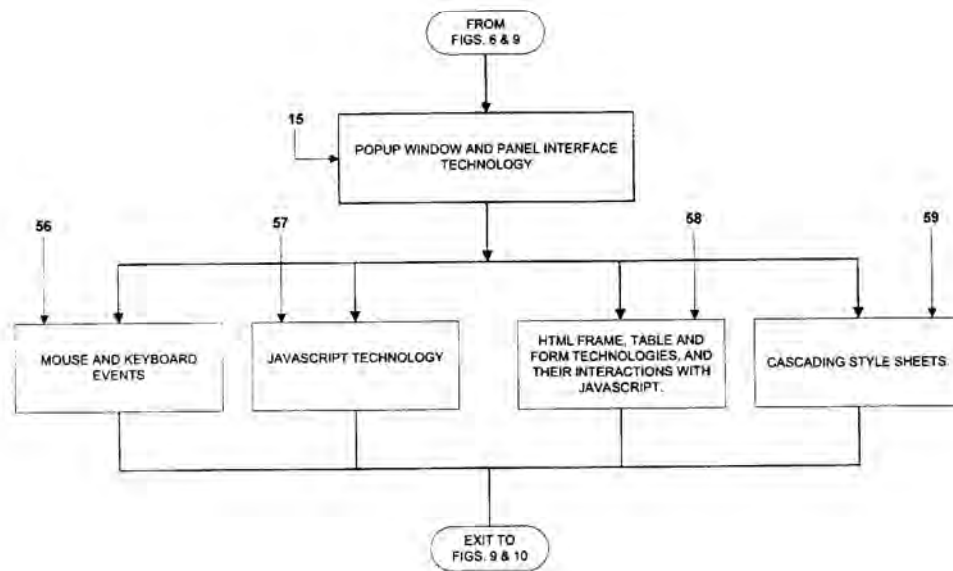
**COMMUNICATION OF USER INPUT DATA AND  
STATUS BETWEEN THE ENGINE AND THE  
INTERFACE.**

**U.S. Patent**

**Apr. 8, 2003**

**Sheet 9 of 68**

**US 6,546,397 B1**



*Fig. 7a*

**POPUP WINDOW AND PANEL INTERFACE  
AND COLOR TECHNOLOGY**

U.S. Patent

Apr. 8, 2003

Sheet 10 of 68

US 6,546,397 B1

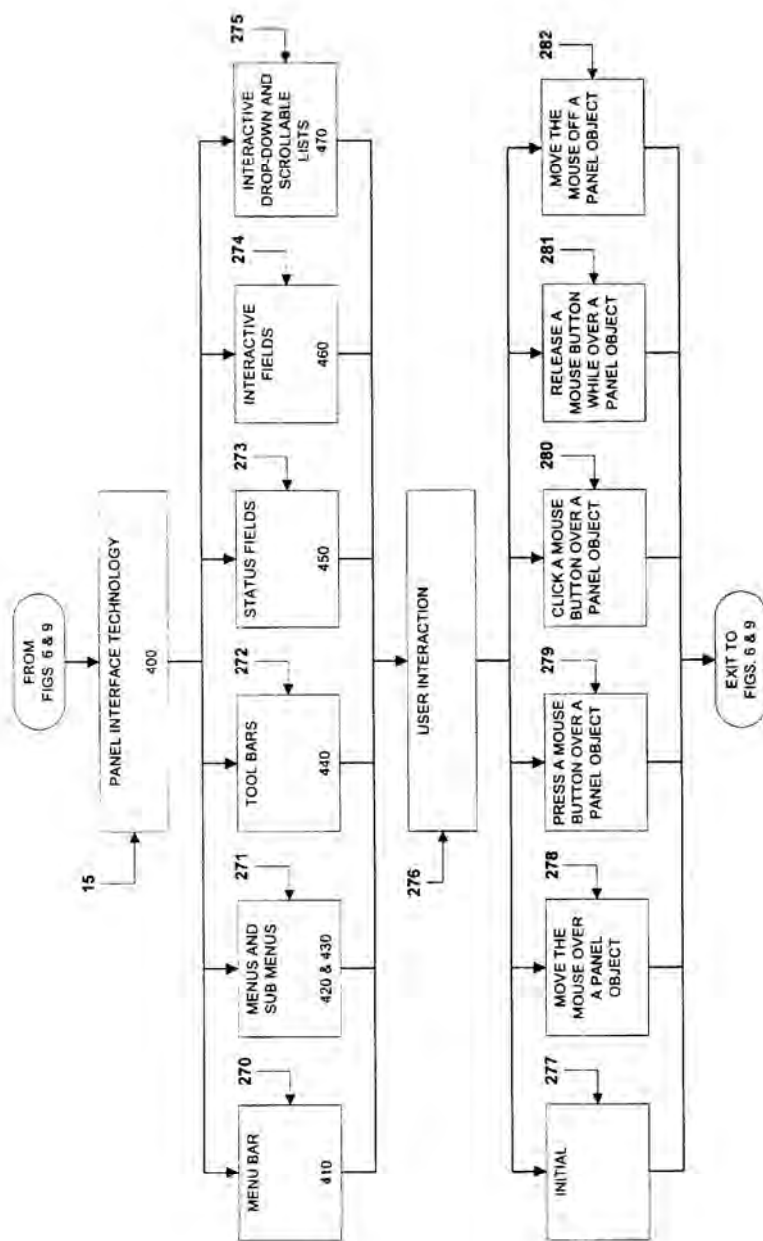


Fig. 7b

IMPLEMENTATION OF PANEL INTERFACE OBJECTS

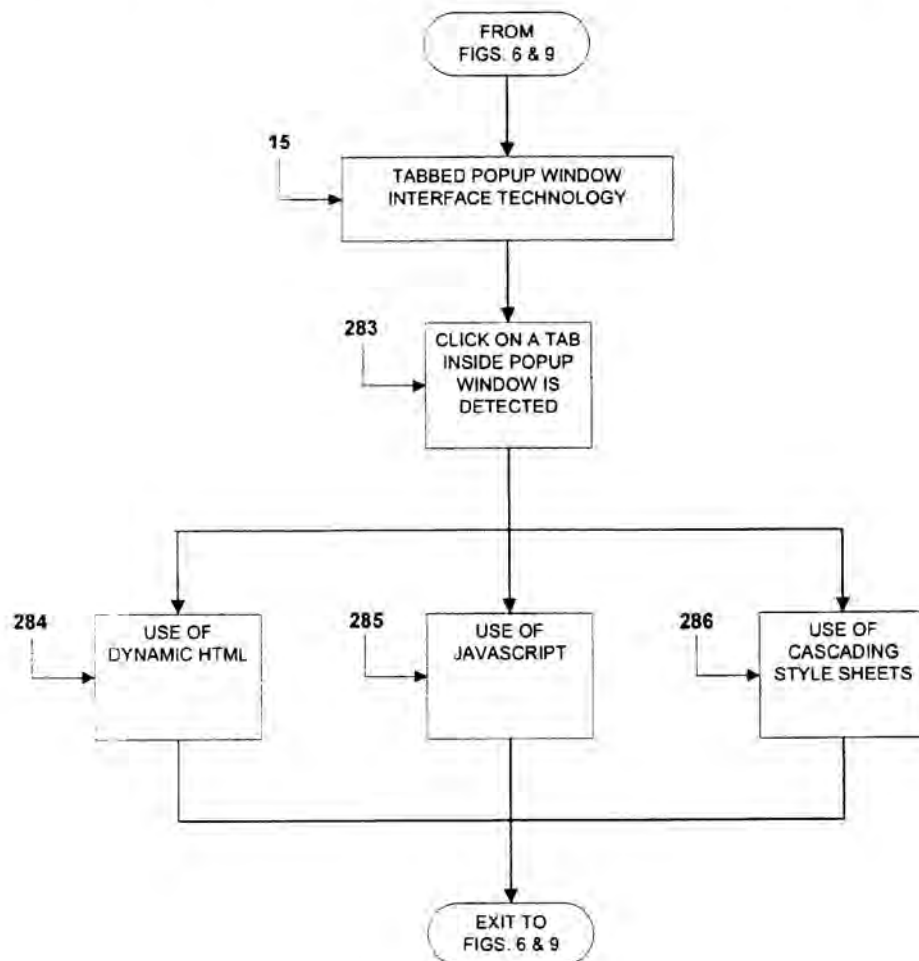


**U.S. Patent**

**Apr. 8, 2003**

**Sheet 11 of 68**

**US 6,546,397 B1**



*Fig. 7c*

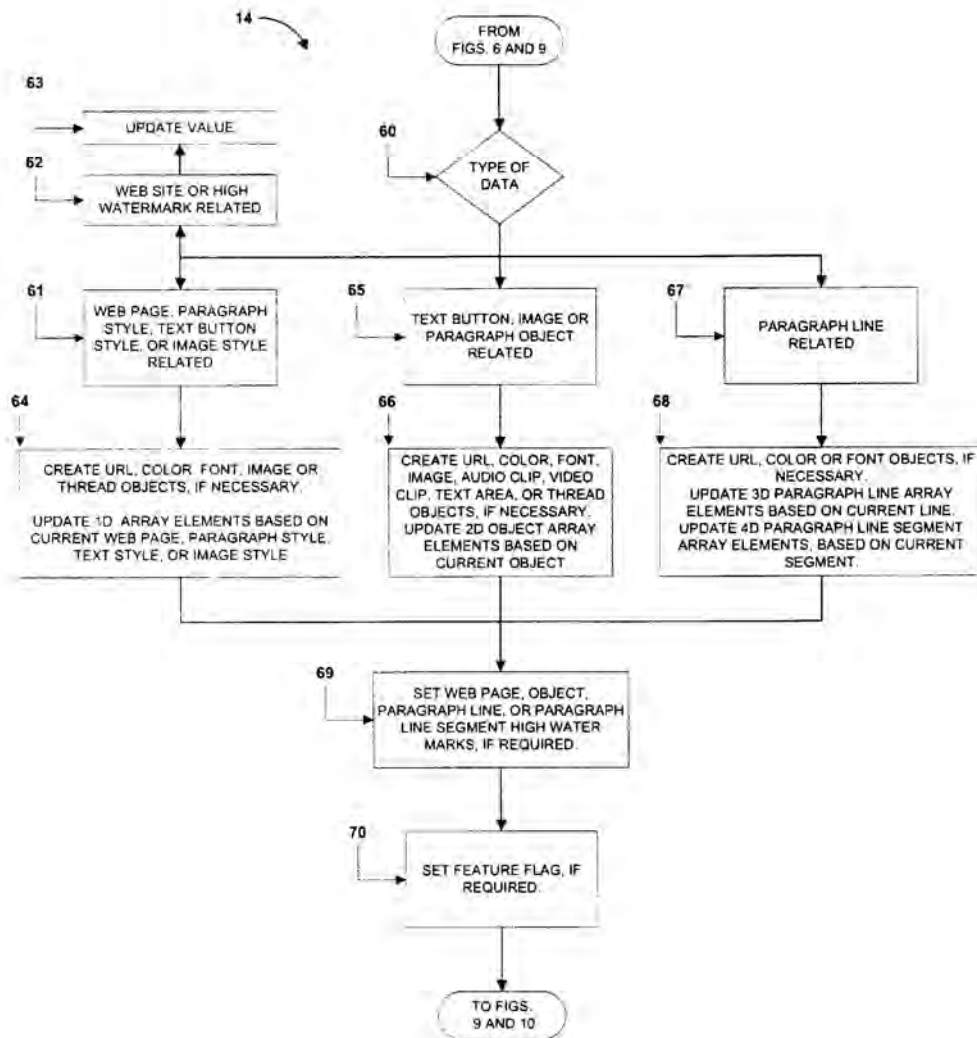
## **IMPLEMENTATION OF TABBED POPUP WINDOWS**

U.S. Patent

Apr. 8, 2003

Sheet 12 of 68

US 6,546,397 B1

*Fig. 8***UPDATE INTERNAL DATA BASE AND SET FEATURE FLAGS**

U.S. Patent

Apr. 8, 2003

Sheet 13 of 68

US 6,546,397 B1

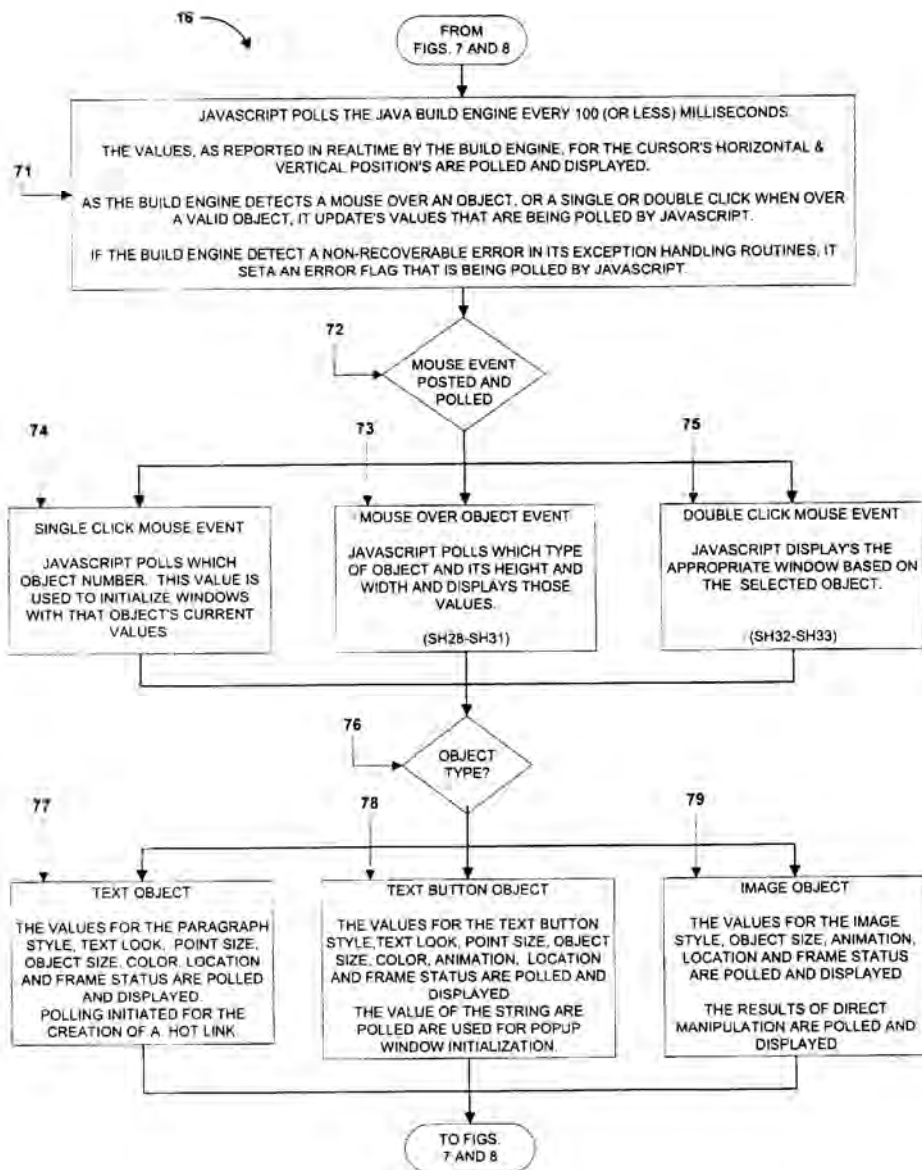


Fig. 9

## POLLING METHODS

U.S. Patent

Apr. 8, 2003

Sheet 14 of 68

US 6,546,397 B1

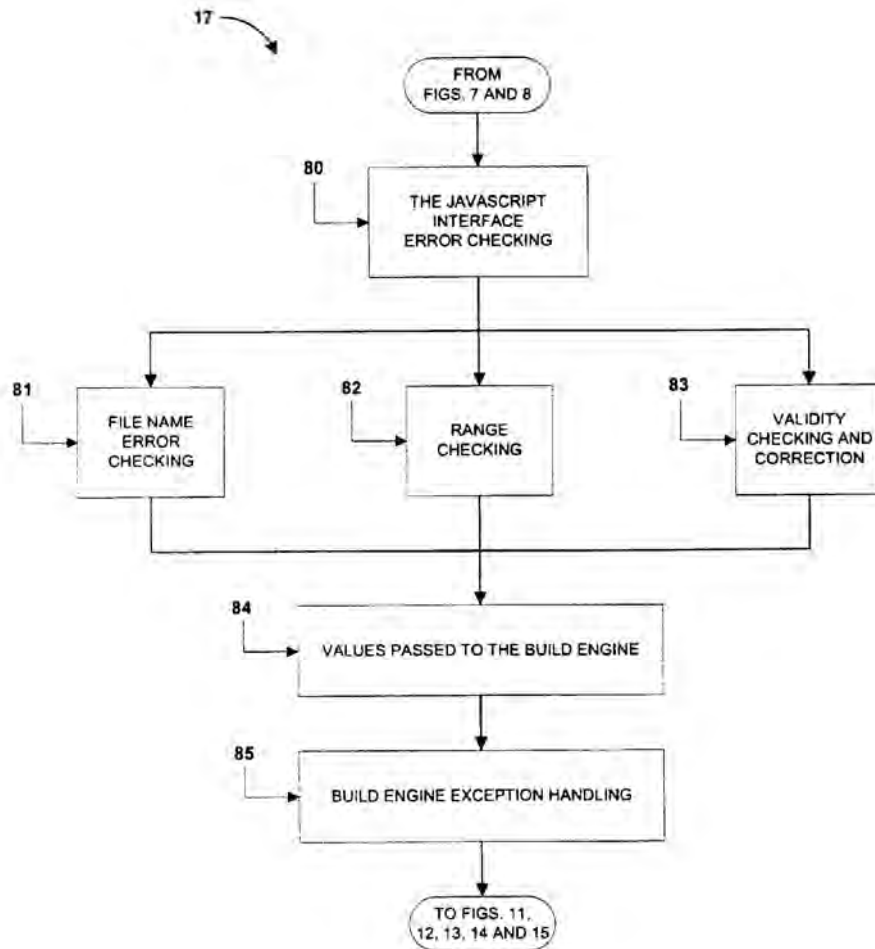


Fig. 10

**ANALYZE INPUT: ERROR CHECKING**

U.S. Patent

Apr. 8, 2003

Sheet 15 of 68

US 6,546,397 B1

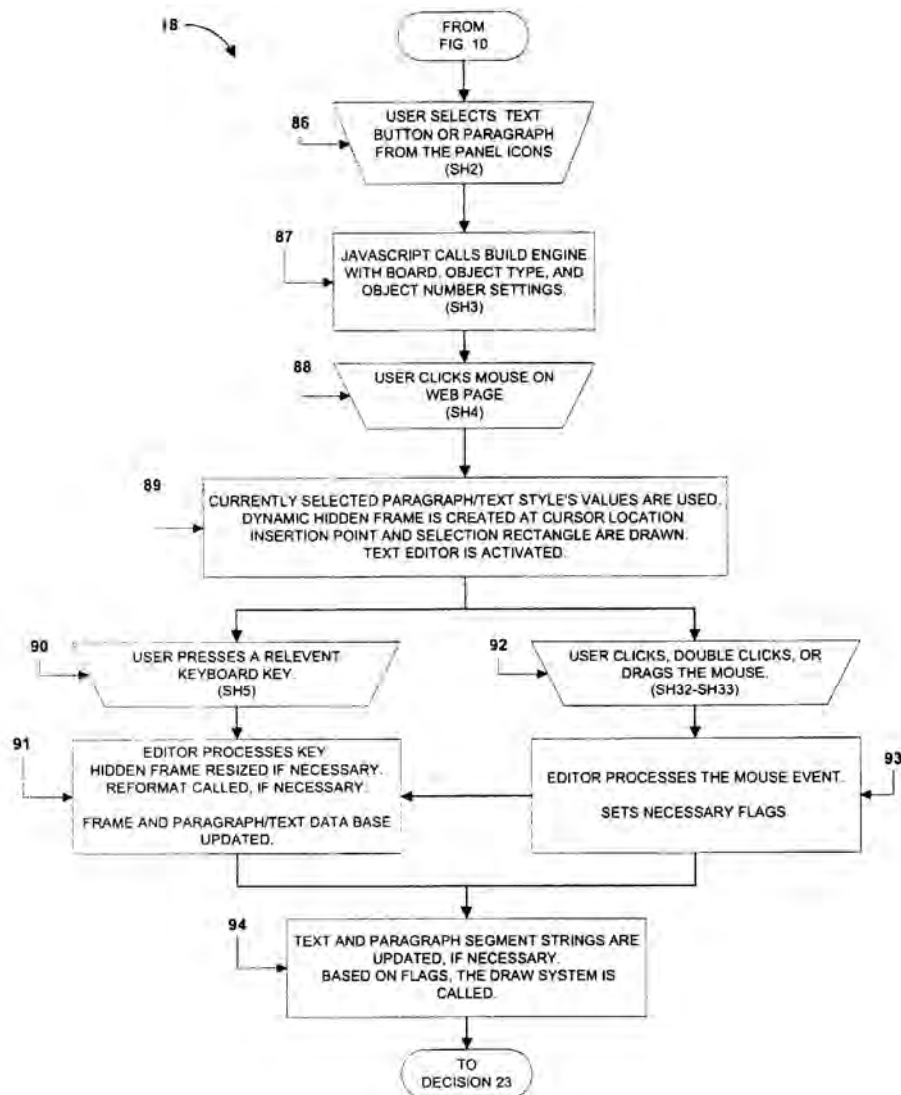


Fig. 11

DIRECT WEB PAGE DATA ENTRY AND TEXT PROCESSING

U.S. Patent

Apr. 8, 2003

Sheet 16 of 68

US 6,546,397 B1

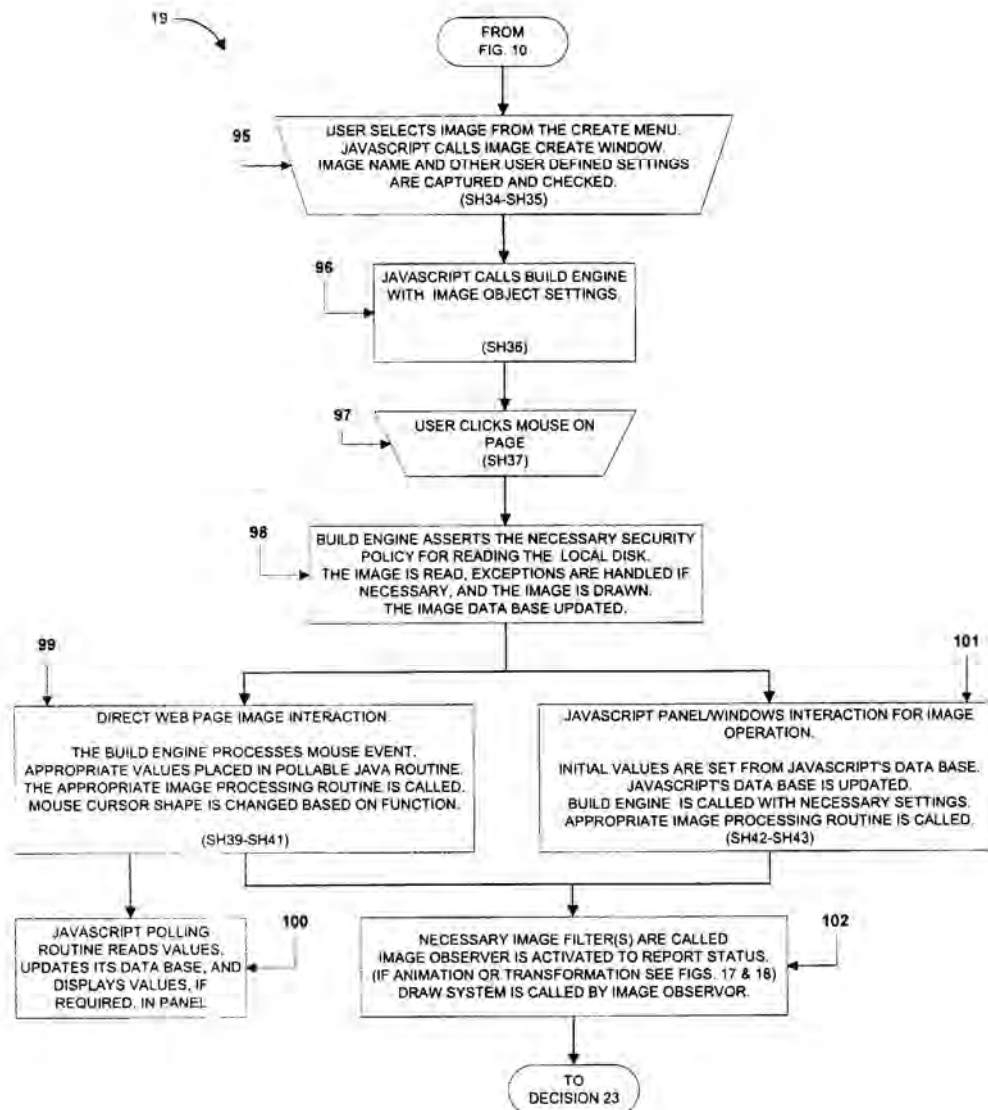


Fig. 12

## IMAGE PROCESSING

U.S. Patent

Apr. 8, 2003

Sheet 17 of 68

US 6,546,397 B1

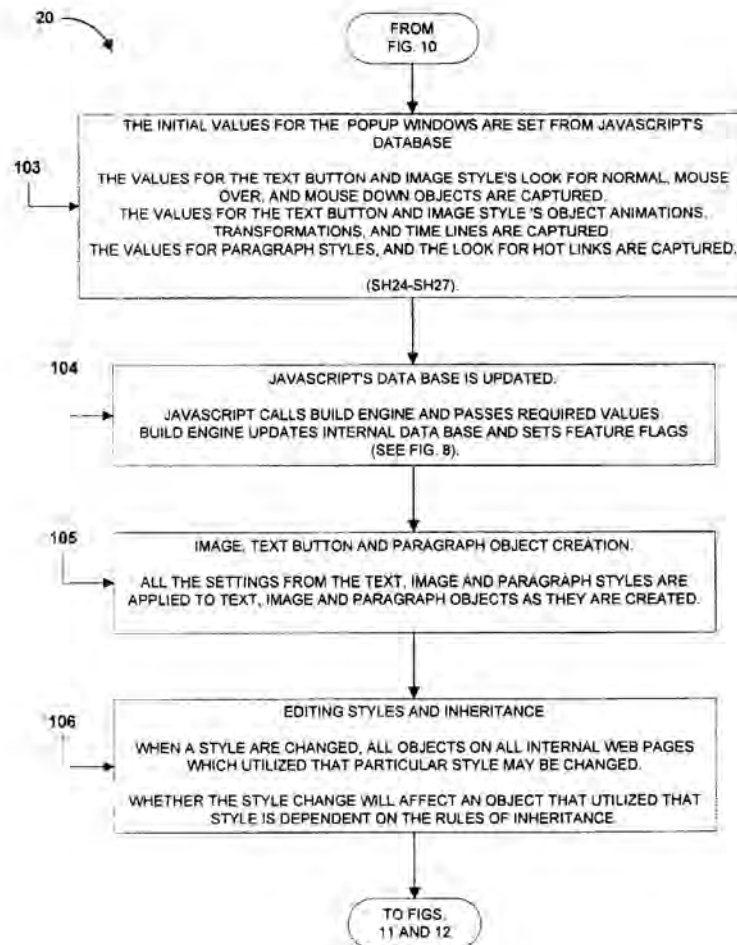


Fig. 13

**BUTTON, IMAGE AND PARAGRAPH STYLE SETTINGS  
AND TECHNOLOGY**

U.S. Patent

Apr. 8, 2003

Sheet 18 of 68

US 6,546,397 B1

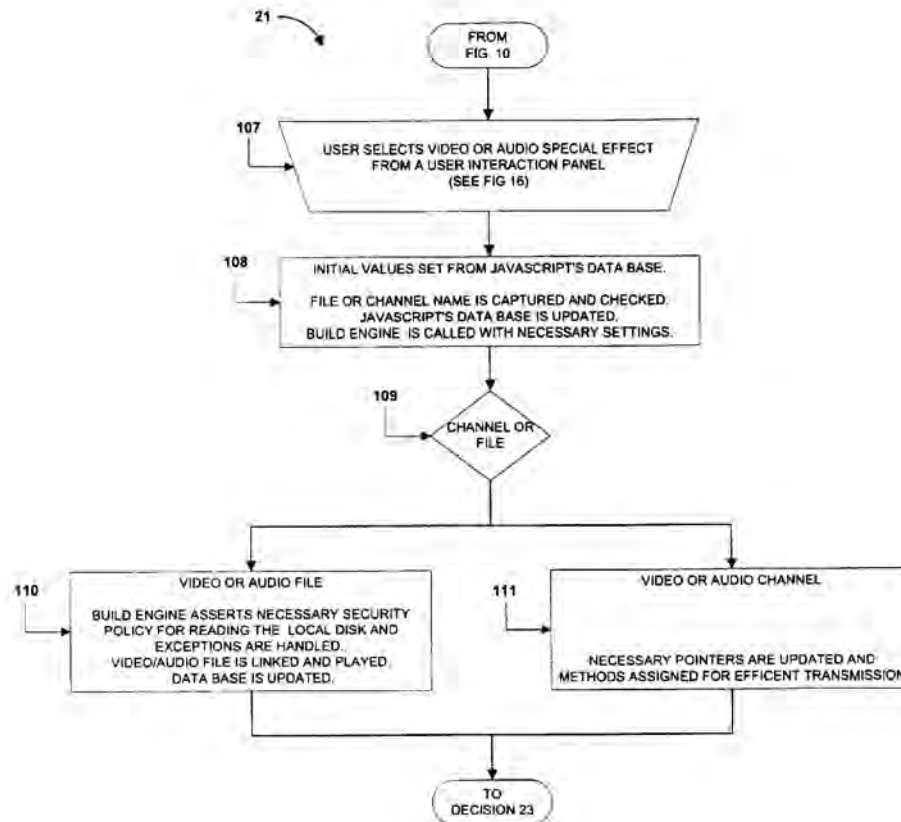


Fig. 14

## VIDEO AND AUDIO FILE/CHANNEL PROCESSING

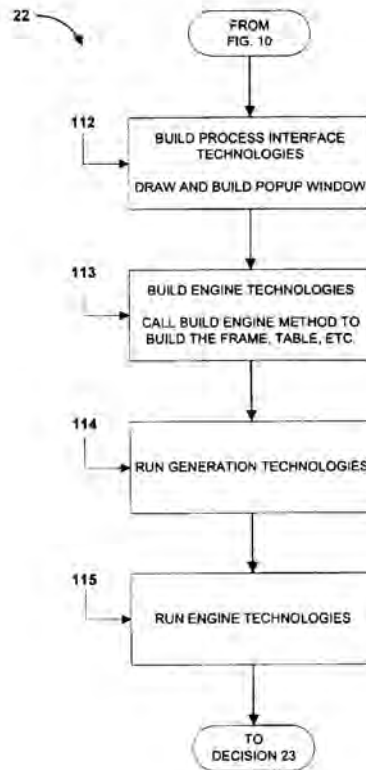


**U.S. Patent**

**Apr. 8, 2003**

**Sheet 19 of 68**

**US 6,546,397 B1**



*Fig. 15*

**FRAMES, TABLES, FORMS AND DRAW OBJECTS**

U.S. Patent

Apr. 8, 2003

Sheet 20 of 68

US 6,546,397 B1

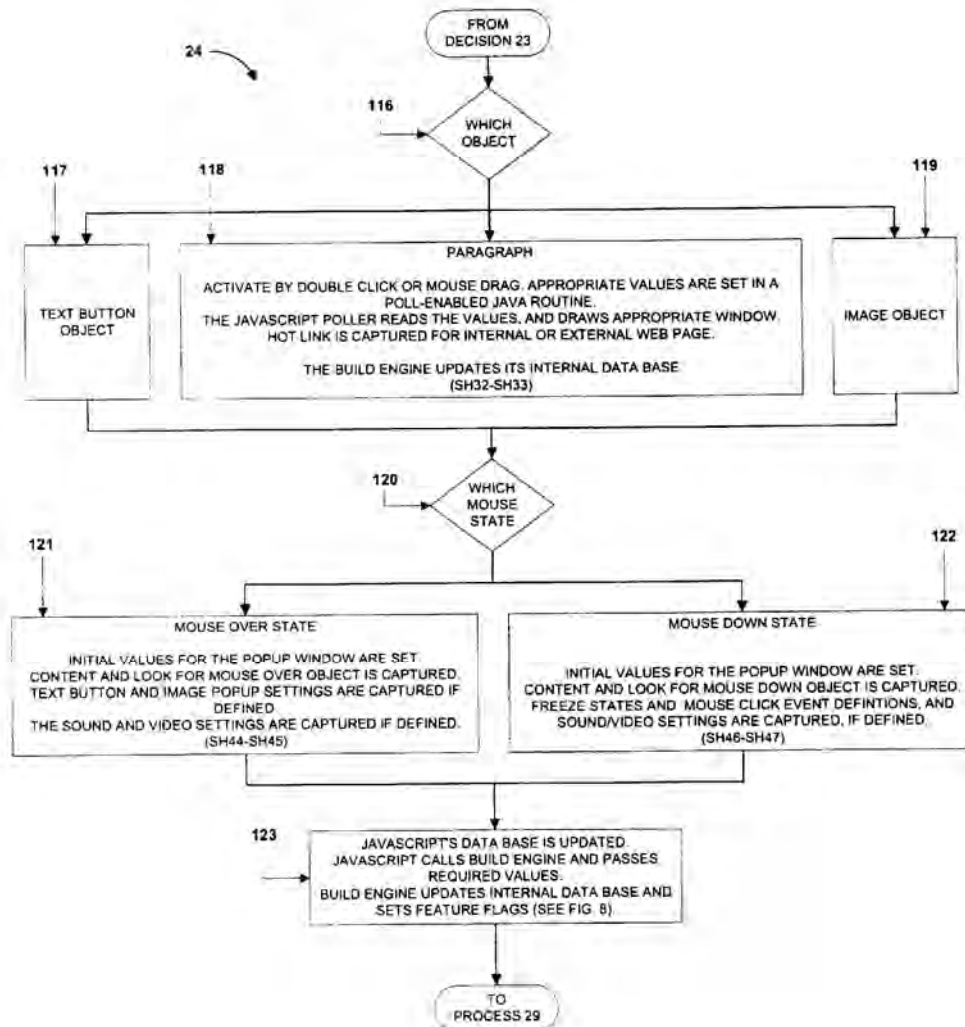


Fig. 16

**USER INTERACTION SETTINGS AND TECHNOLOGY**

U.S. Patent

Apr. 8, 2003

Sheet 21 of 68

US 6,546,397 B1

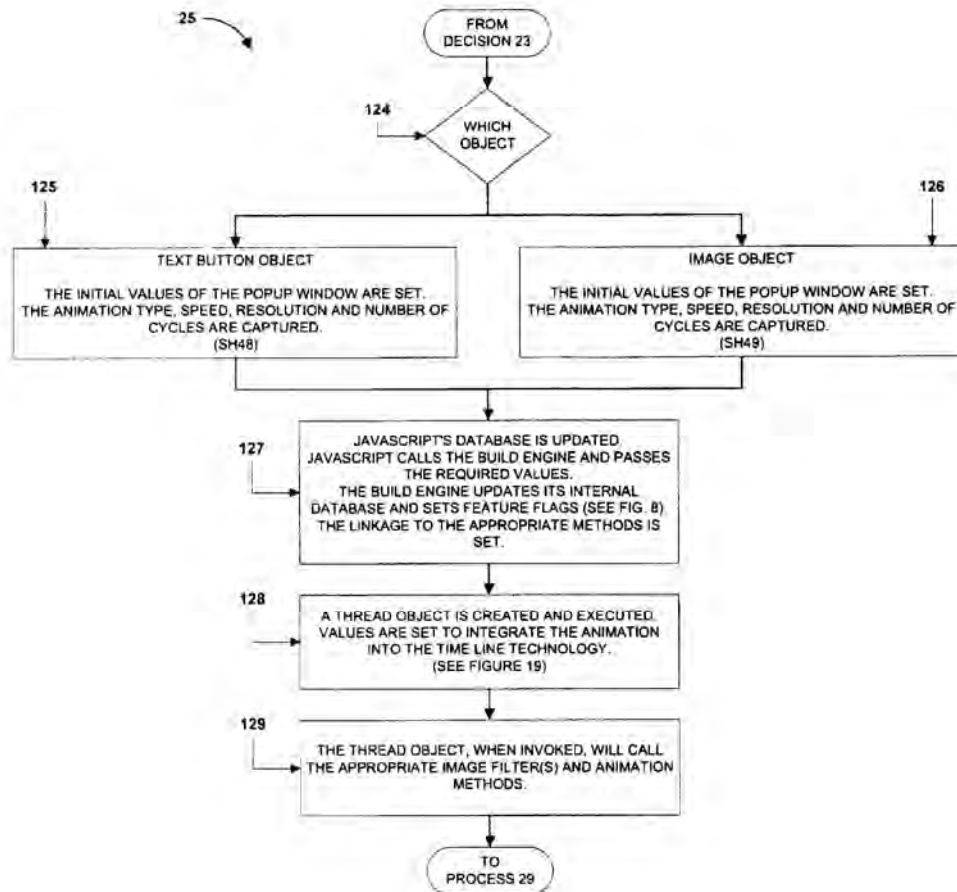


Fig. 17

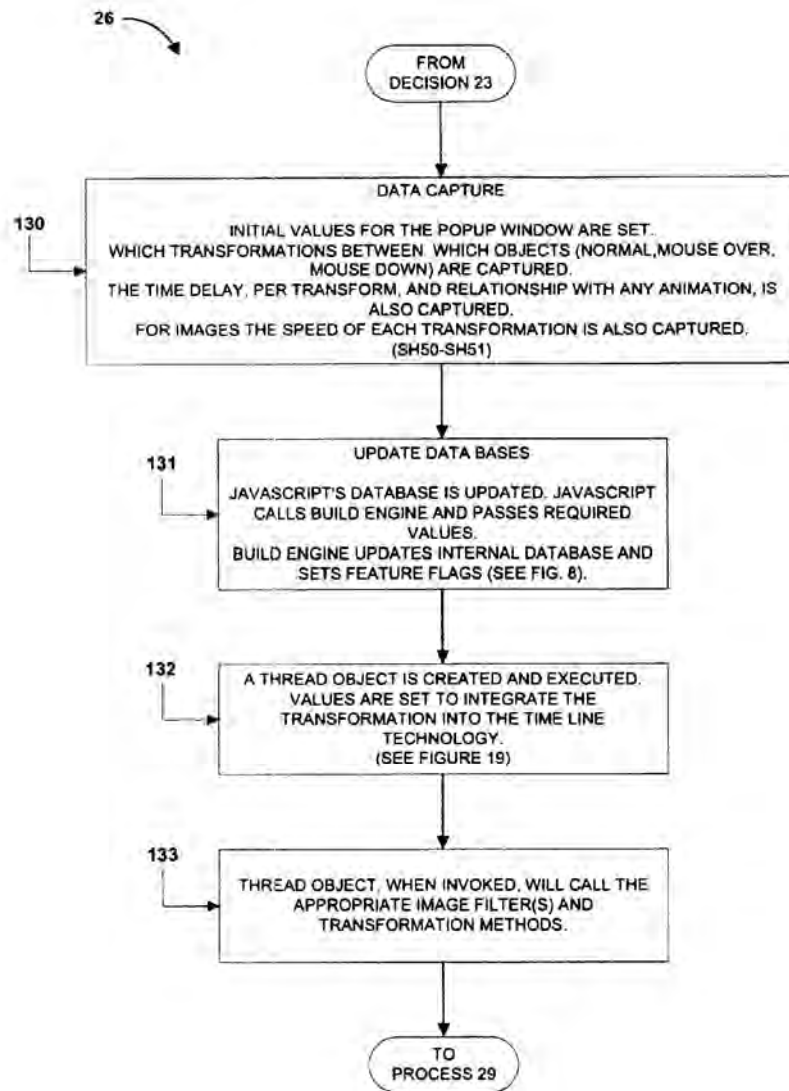
# ANIMATION SETTINGS AND TECHNOLOGY

U.S. Patent

Apr. 8, 2003

Sheet 22 of 68

US 6,546,397 B1

*Fig. 18*

TRANSFORMATION SETTINGS AND TECHNOLOGY
--

U.S. Patent

Apr. 8, 2003

Sheet 23 of 68

US 6,546,397 B1

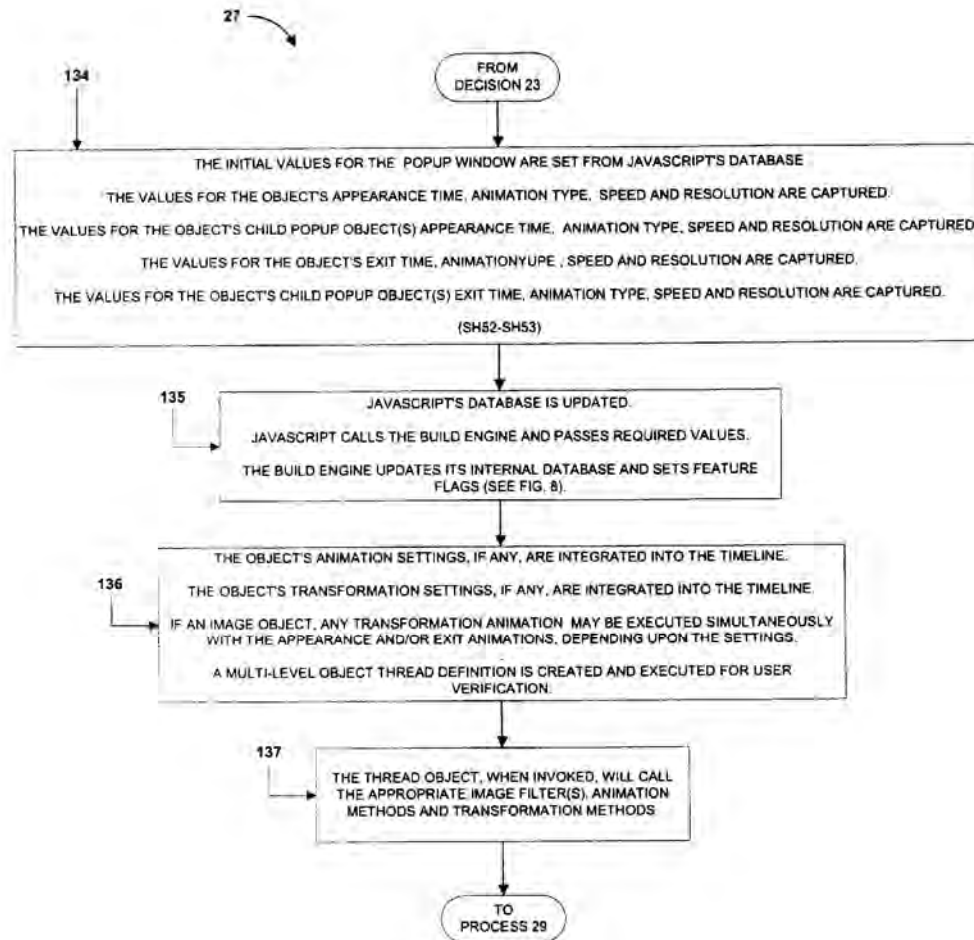


Fig. 19

OBJECT TIME LINES AND TECHNOLOGY

U.S. Patent

Apr. 8, 2003

Sheet 24 of 68

US 6,546,397 B1

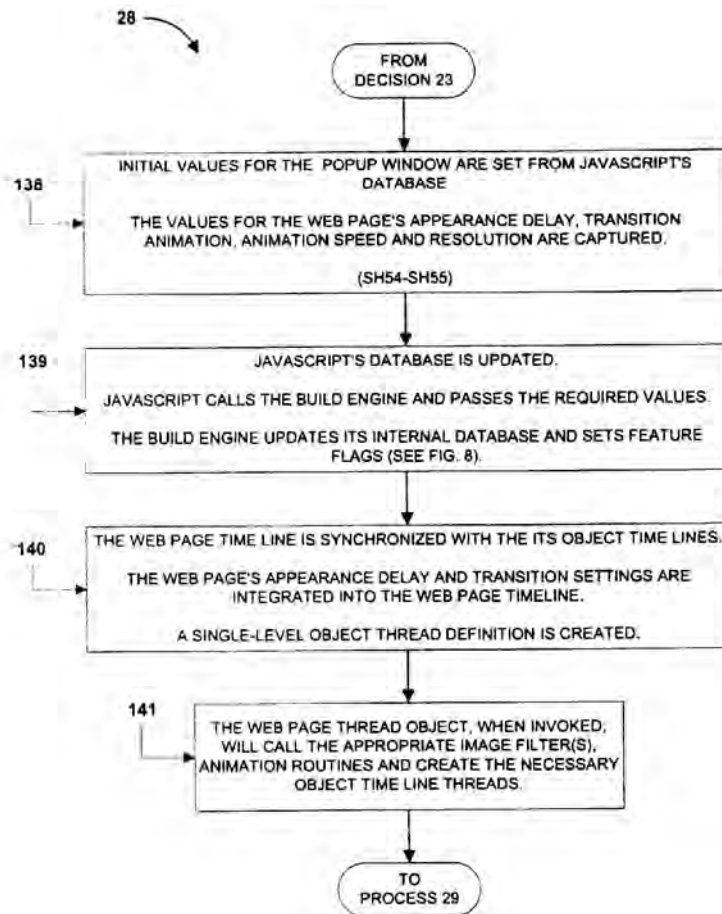


Fig. 20

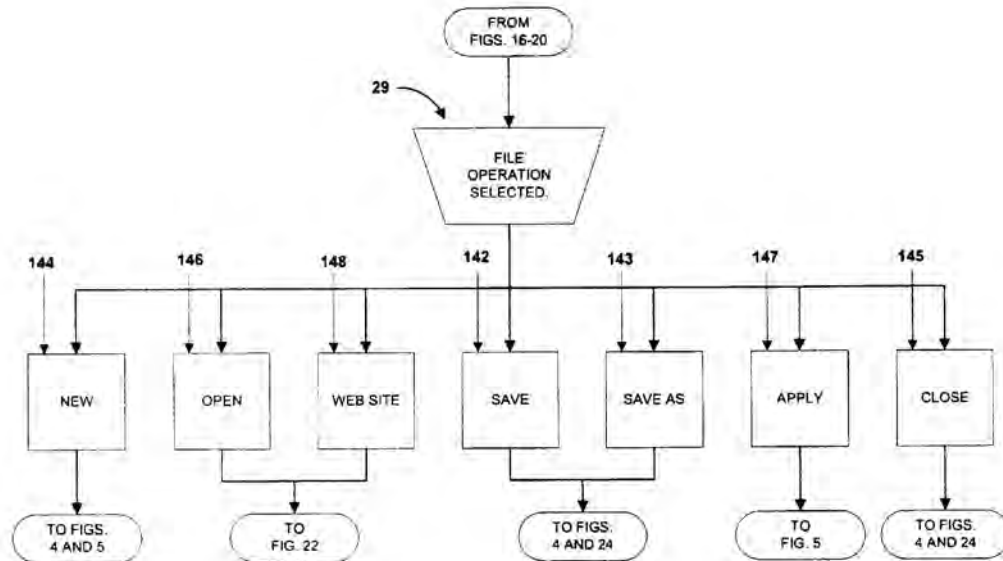
**WEB PAGE TRANSITION ANIMATIONS, TIME LINE  
SETTINGS AND TECHNOLOGY**

**U.S. Patent**

**Apr. 8, 2003**

**Sheet 25 of 68**

**US 6,546,397 B1**



*fig. 21a*

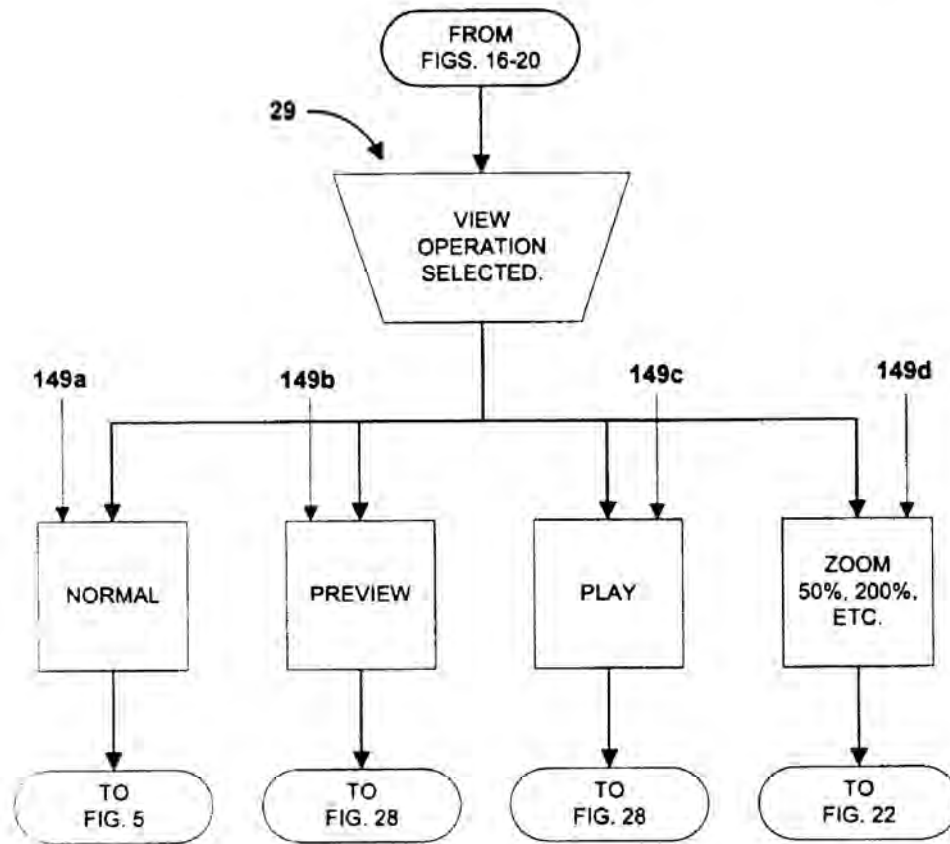
## **FILE OPERATIONS**

U.S. Patent

Apr. 8, 2003

Sheet 26 of 68

US 6,546,397 B1



*fig. 21b*

## VIEW OPERATIONS



U.S. Patent

Apr. 8, 2003

Sheet 27 of 68

US 6,546,397 B1

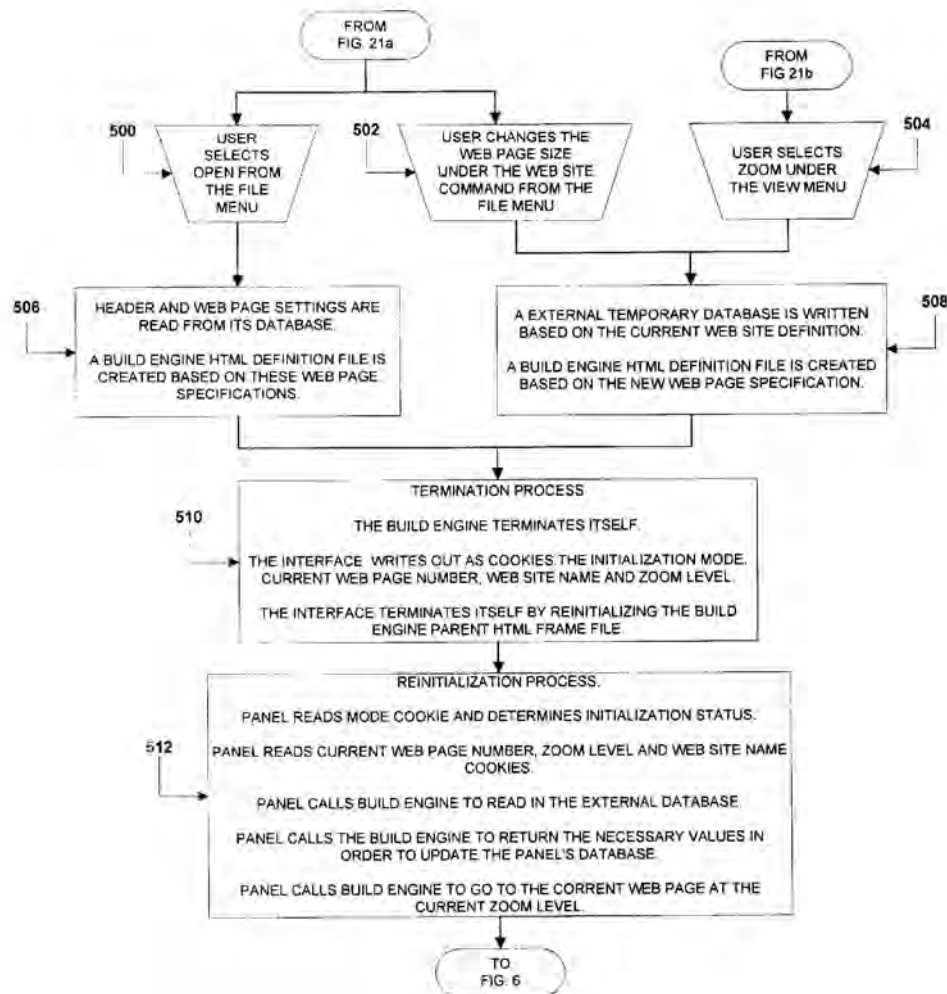
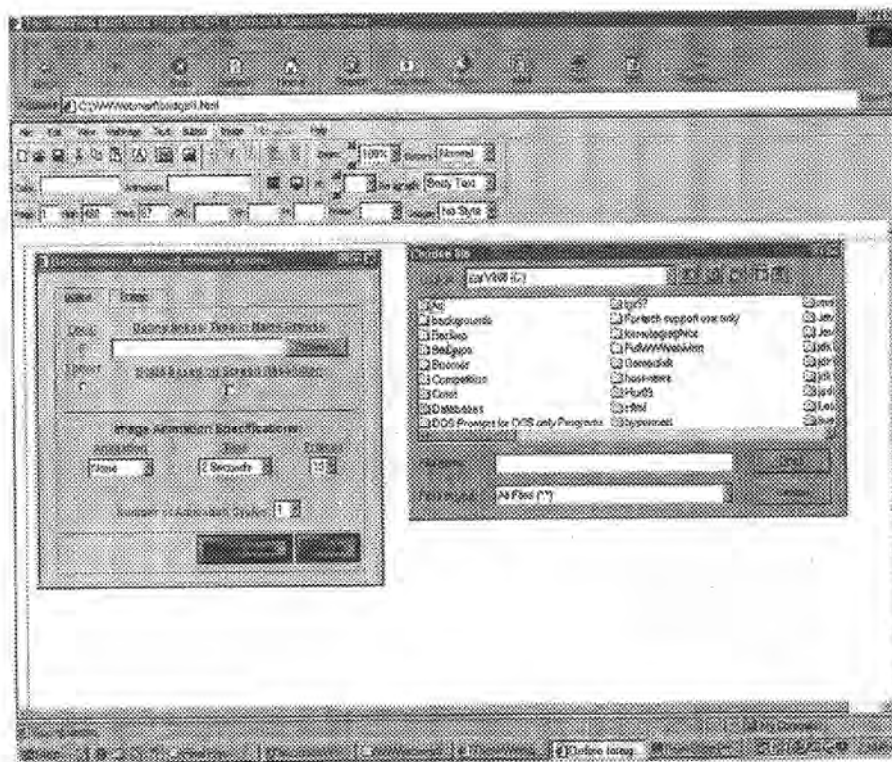


Fig. 22

**DYNAMIC WEB PAGE RESIZING PROCESS**

Fig. 23



U.S. Patent

Apr. 8, 2003

Sheet 29 of 68

US 6,546,397 B1

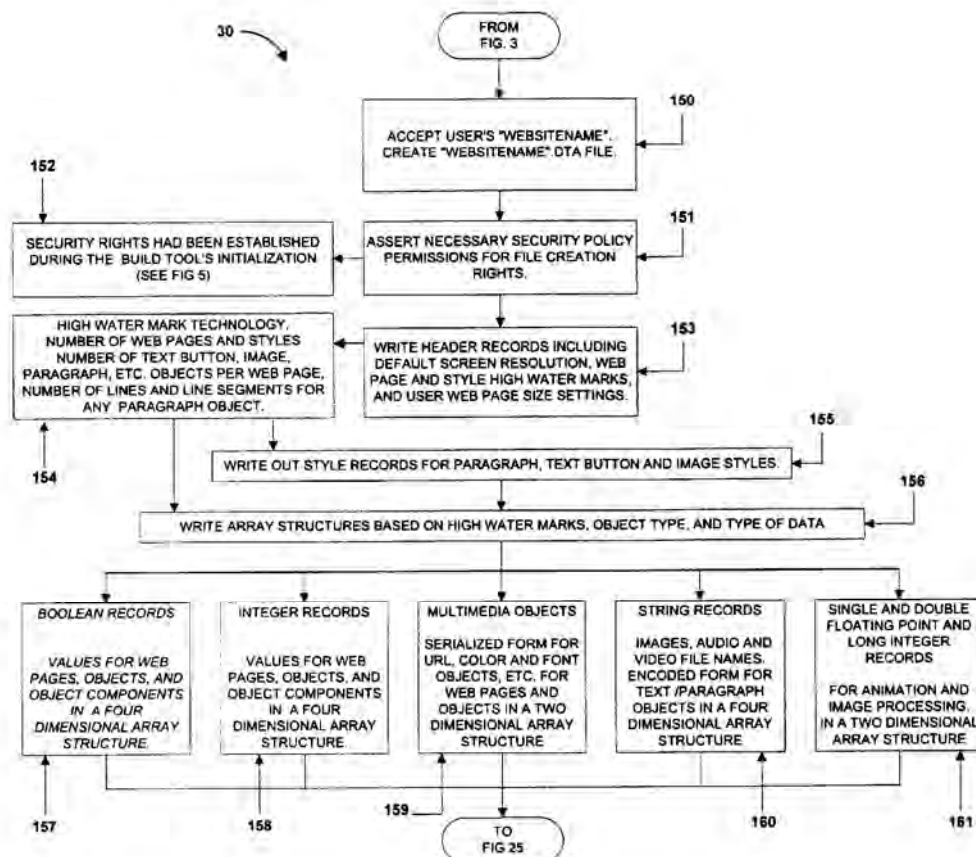


Fig. 24

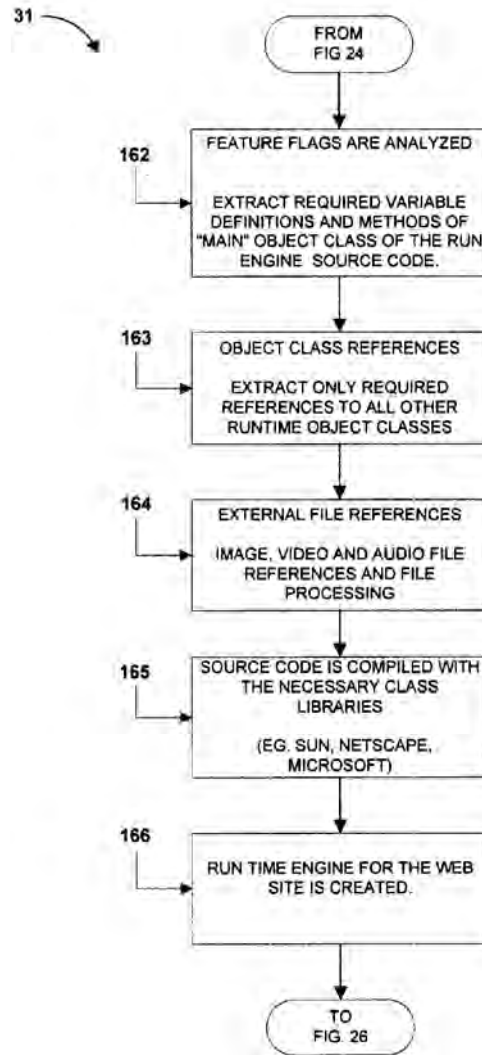
## EXTERNAL DATA BASE CREATION: SECURITY AND OPTIMIZATION TECHNIQUES

U.S. Patent

Apr. 8, 2003

Sheet 30 of 68

US 6,546,397 B1

*Fig. 25*

**CREATE CUSTOMIZED AND OPTIMIZED  
RUNTIME ENGINE**

U.S. Patent

Apr. 8, 2003

Sheet 31 of 68

US 6,546,397 B1

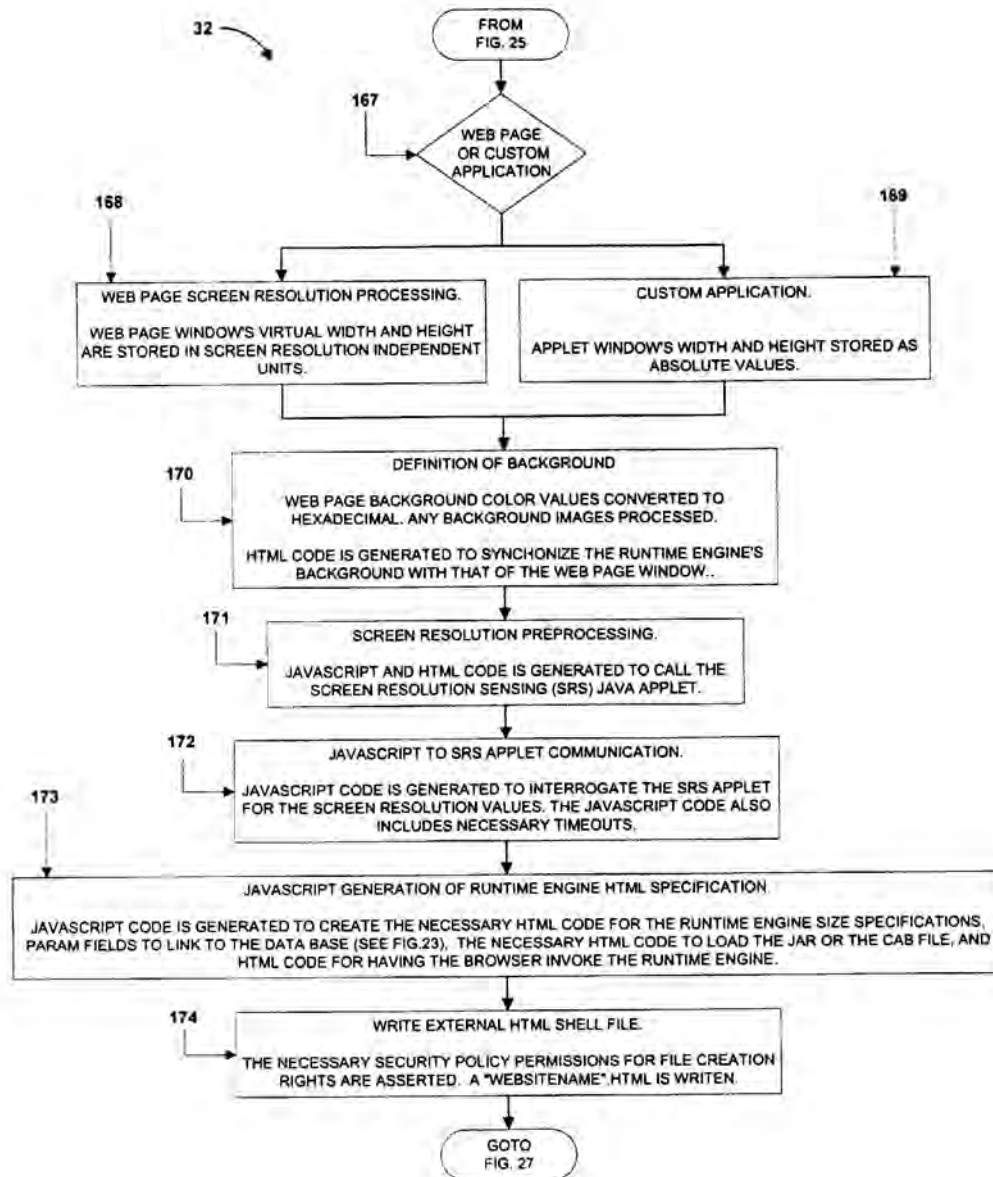


Fig. 26

**CREATE THE HTML SHELL FILE**

U.S. Patent

Apr. 8, 2003

Sheet 32 of 68

US 6,546,397 B1

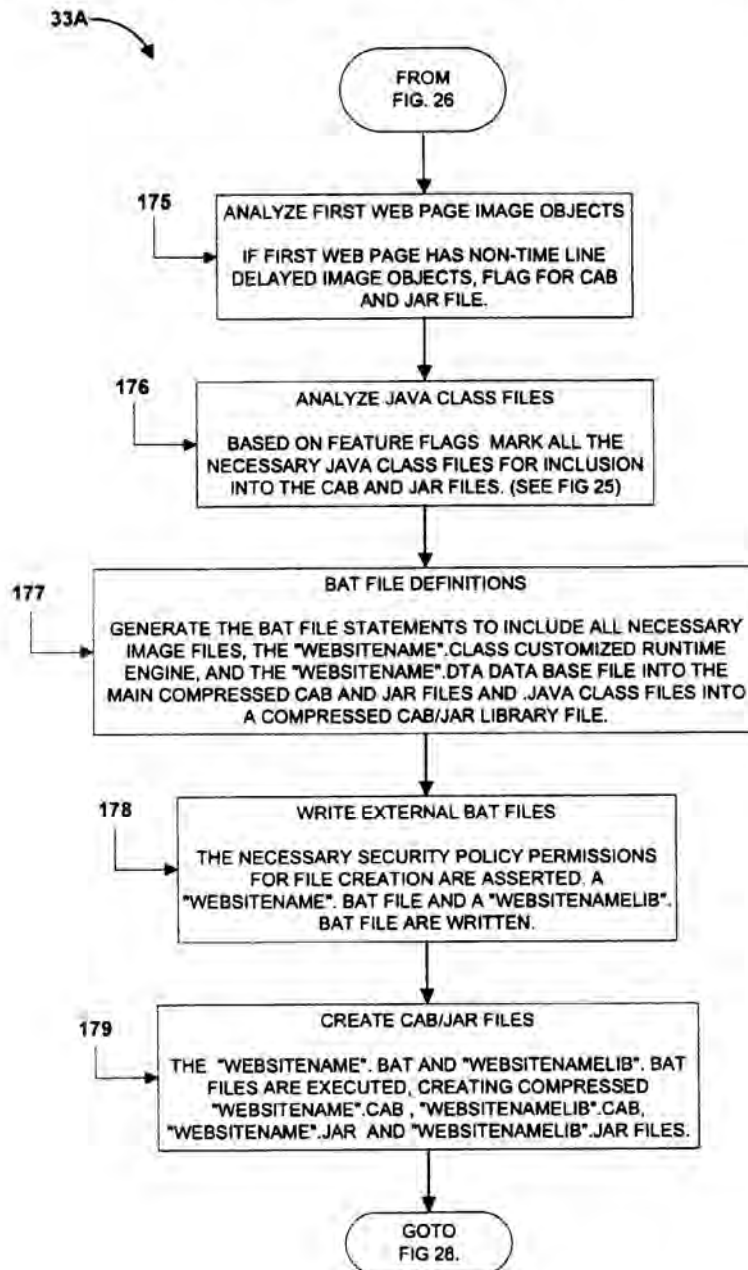


Fig. 27

**CREATE THE CAB/JAR FILES**

U.S. Patent

Apr. 8, 2003

Sheet 33 of 68

US 6,546,397 B1

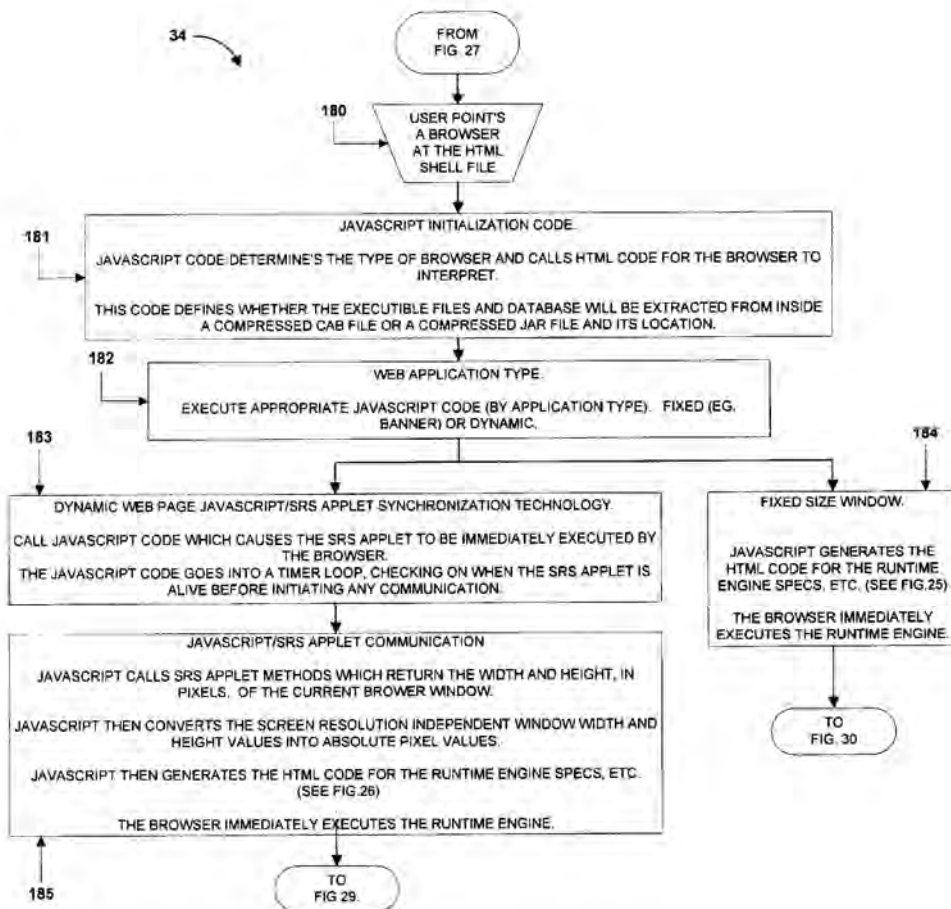


Fig. 28

**WEB PAGE SIZE GENERATION TECHNOLOGY**

U.S. Patent

Apr. 8, 2003

Sheet 34 of 68

US 6,546,397 B1

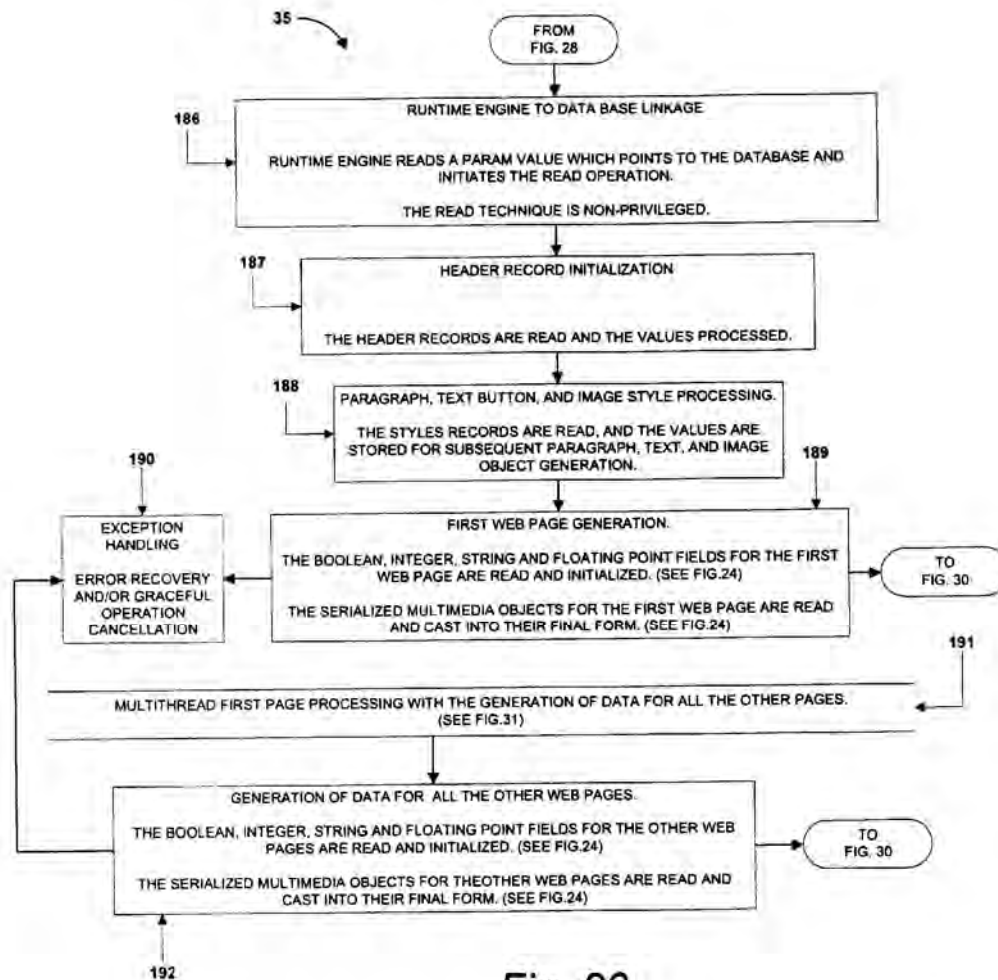


Fig. 29

**READ DATA BASE AND GENERATE  
NECESSARY OBJECTS.**



U.S. Patent

Apr. 8, 2003

Sheet 35 of 68

US 6,546,397 B1

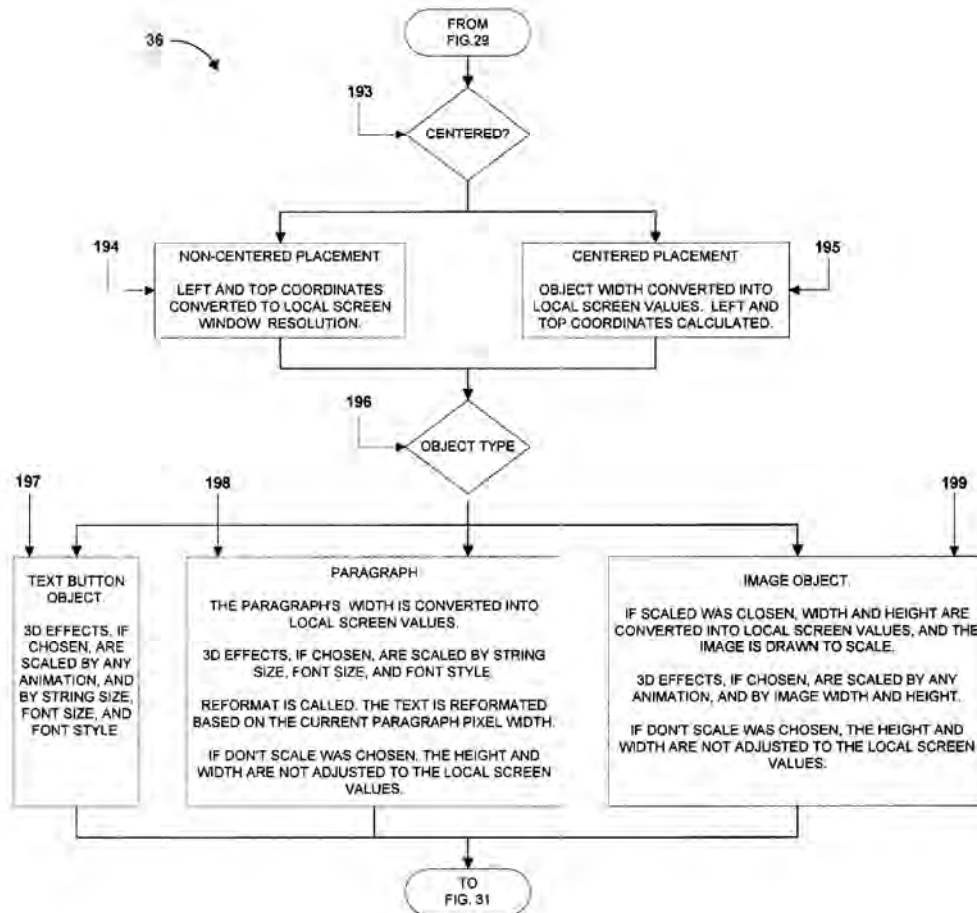


Fig. 30

## WEB PAGE GENERATION WITH SCALING TECHNOLOGY

U.S. Patent

Apr. 8, 2003

Sheet 36 of 68

US 6,546,397 B1

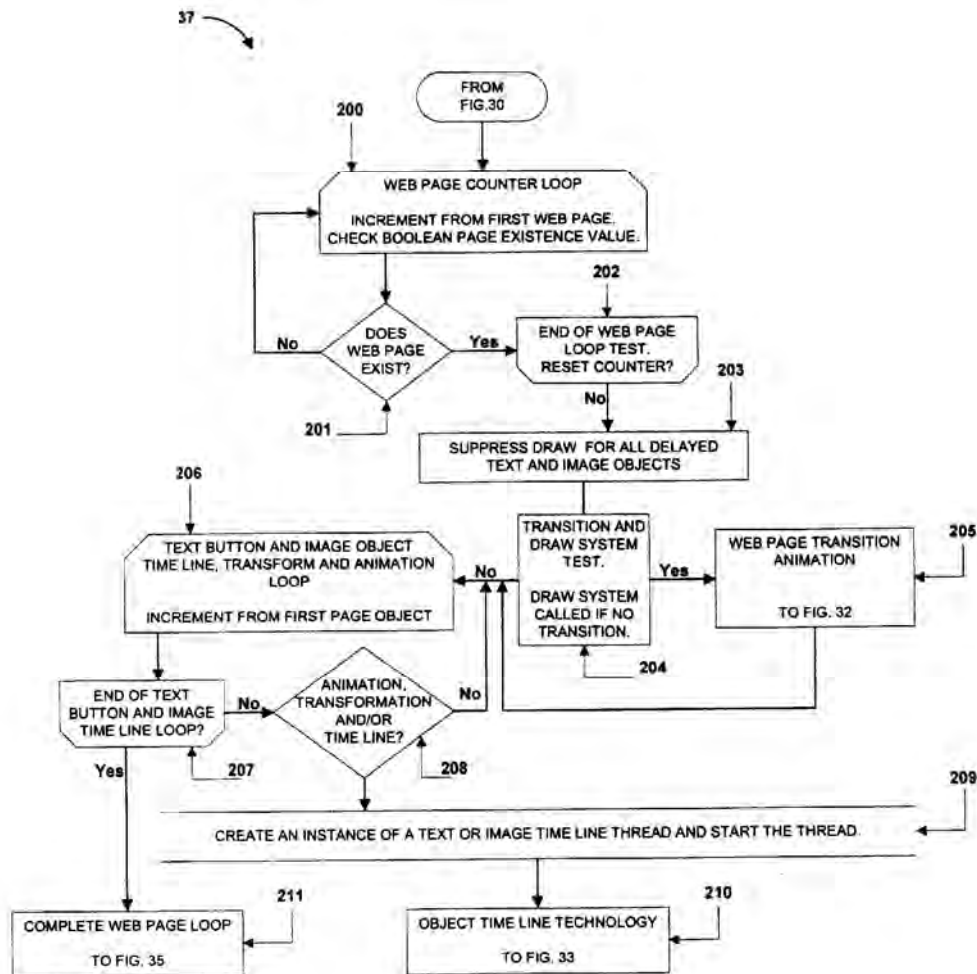


Fig. 31

## THE MULTILEVEL WEB PAGE AND OBJECT THREAD TECHNOLOGY

U.S. Patent

Apr. 8, 2003

Sheet 37 of 68

US 6,546,397 B1

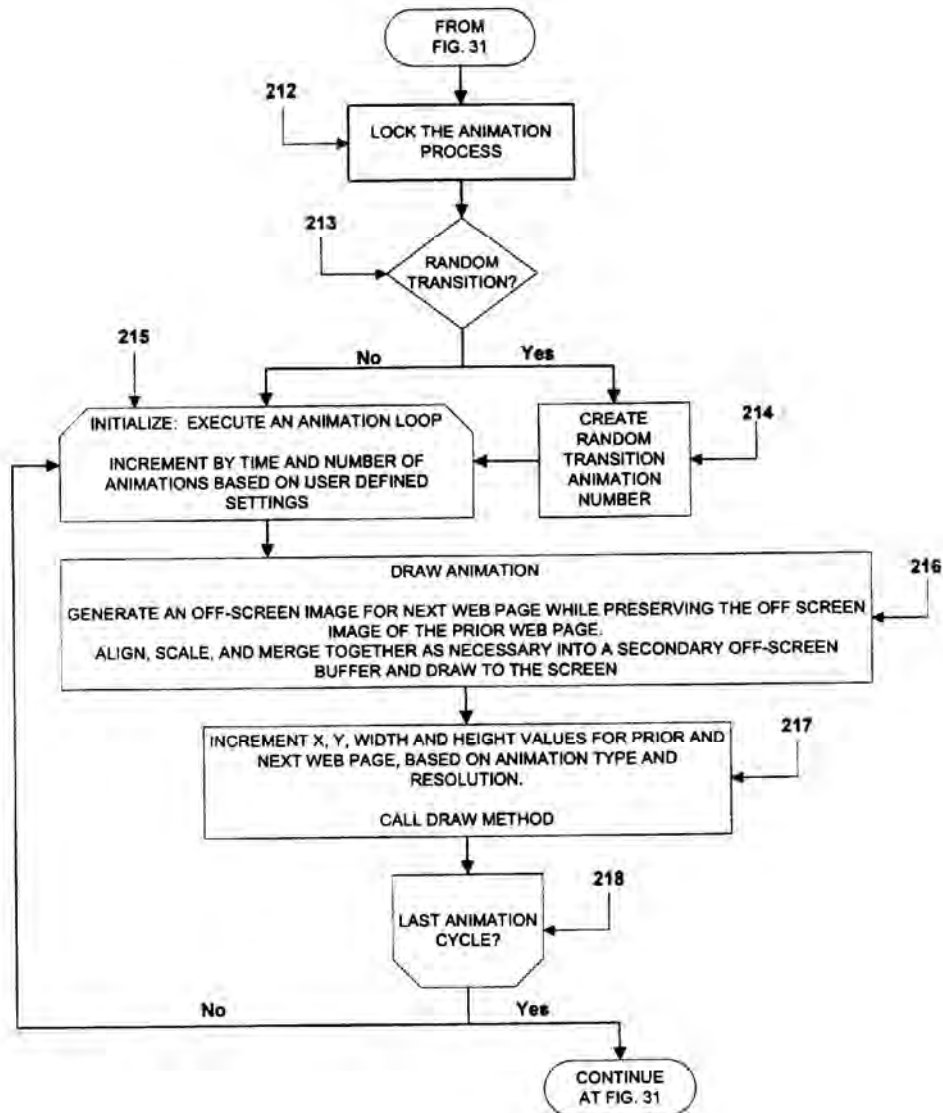


Fig. 32

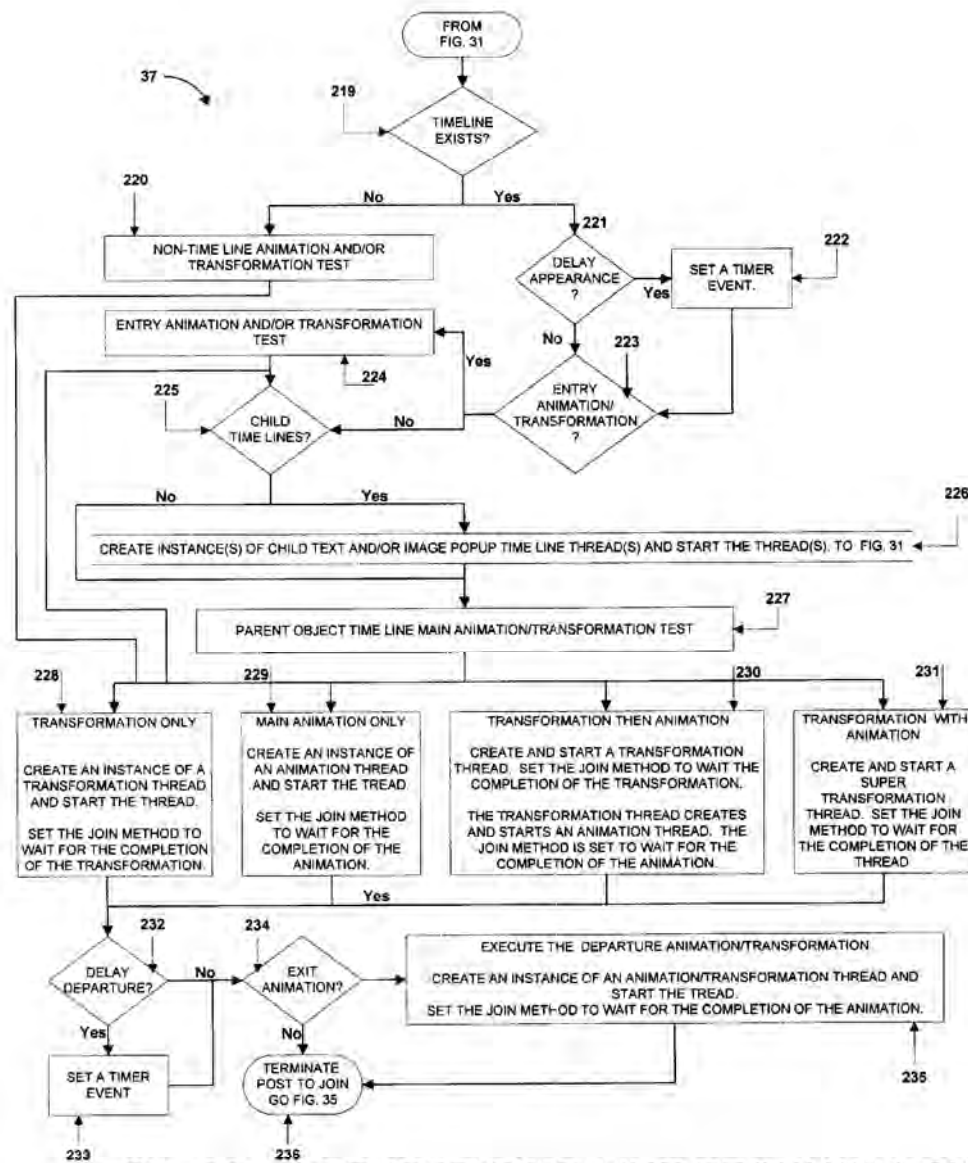
## WEB PAGE TRANSITION ANIMATION

U.S. Patent

Apr. 8, 2003

Sheet 38 of 68

US 6,546,397 B1

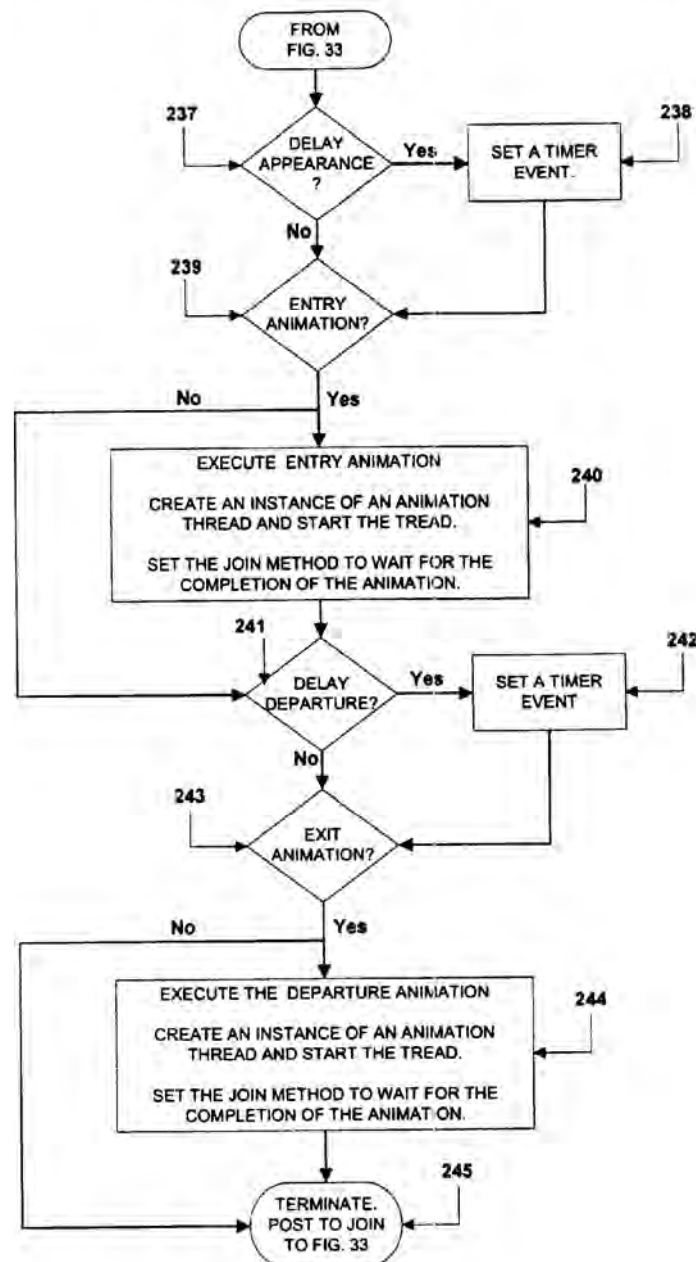


U.S. Patent

Apr. 8, 2003

Sheet 39 of 68

US 6,546,397 B1

*Fig. 34*

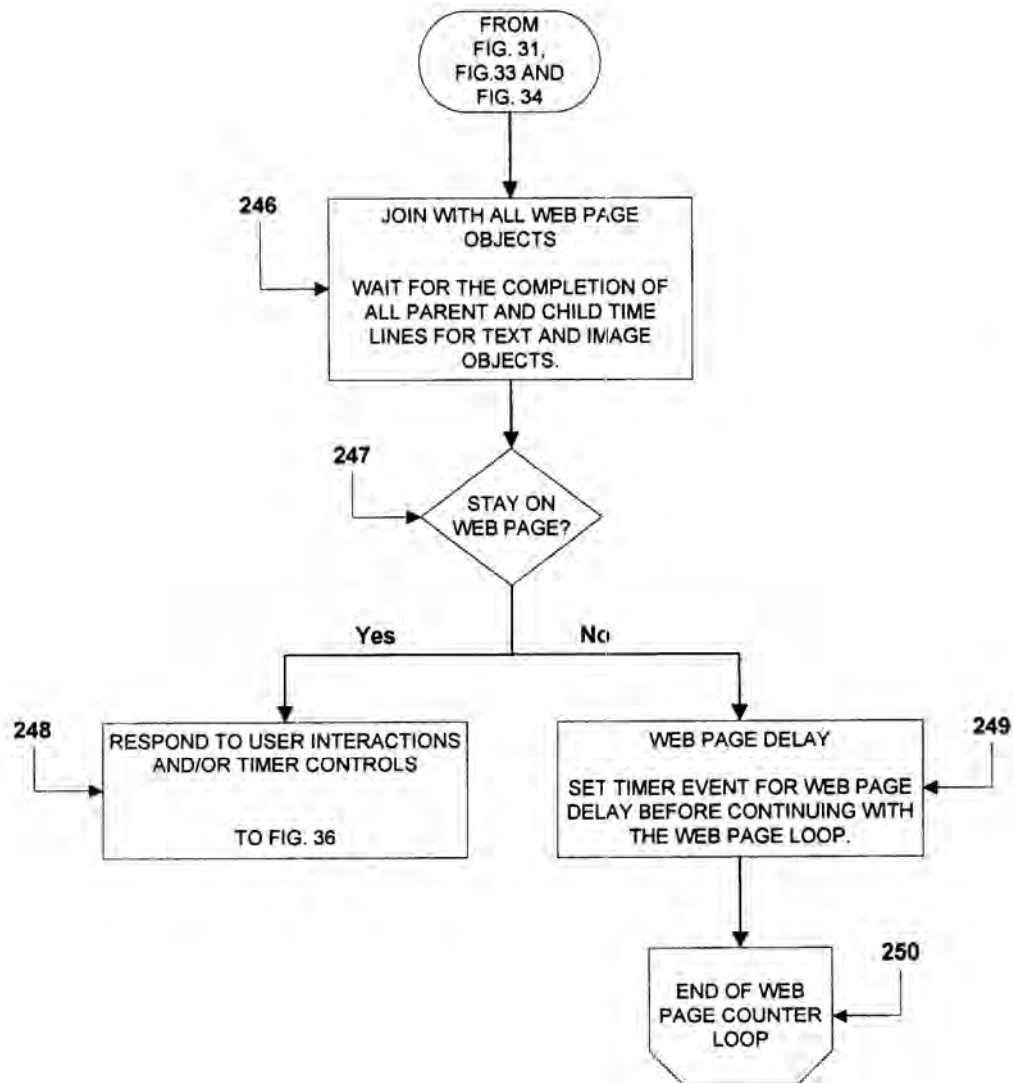
## CHILD TIME LINES FOR TEXT BUTTON AND IMAGE OBJECTS

U.S. Patent

Apr. 8, 2003

Sheet 40 of 68

US 6,546,397 B1

*Fig. 35*

## COMPLETE WEB PAGE AND OBJECT THREAD LOOP

U.S. Patent

Apr. 8, 2003

Sheet 41 of 68

US 6,546,397 B1

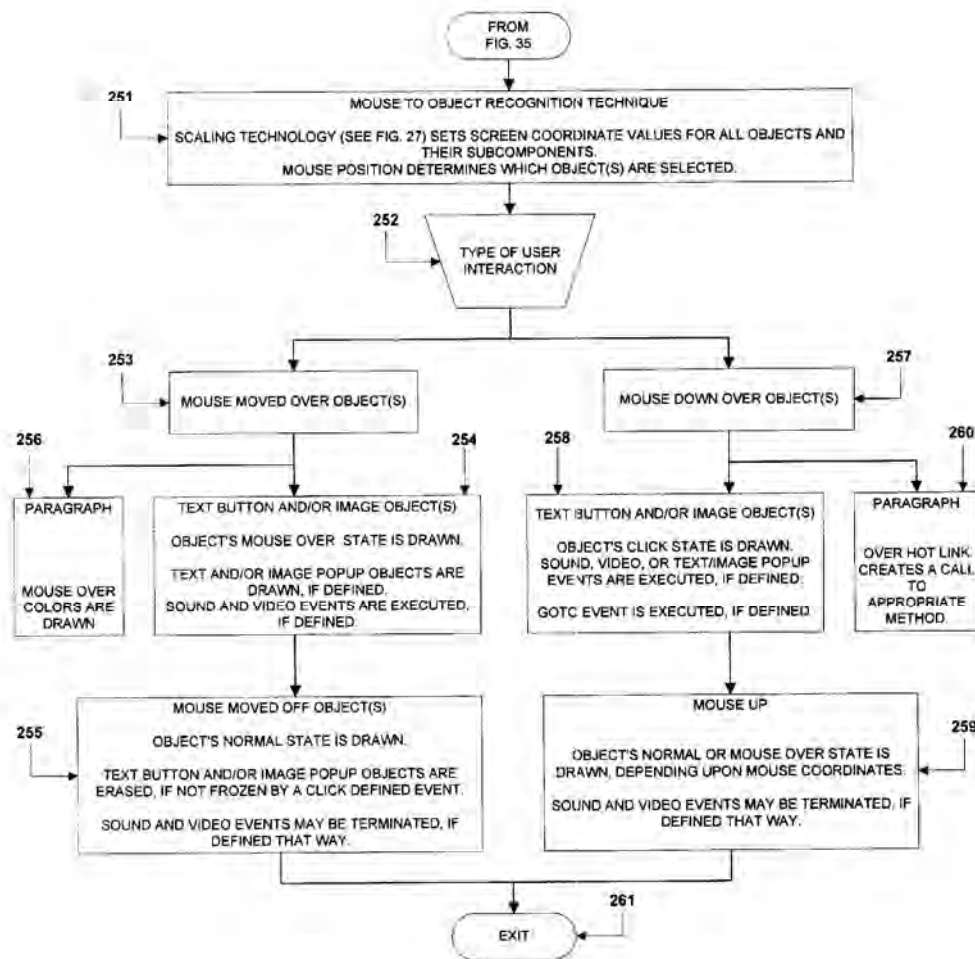


Fig. 36

**RESPOND TO USER INTERACTIONS**

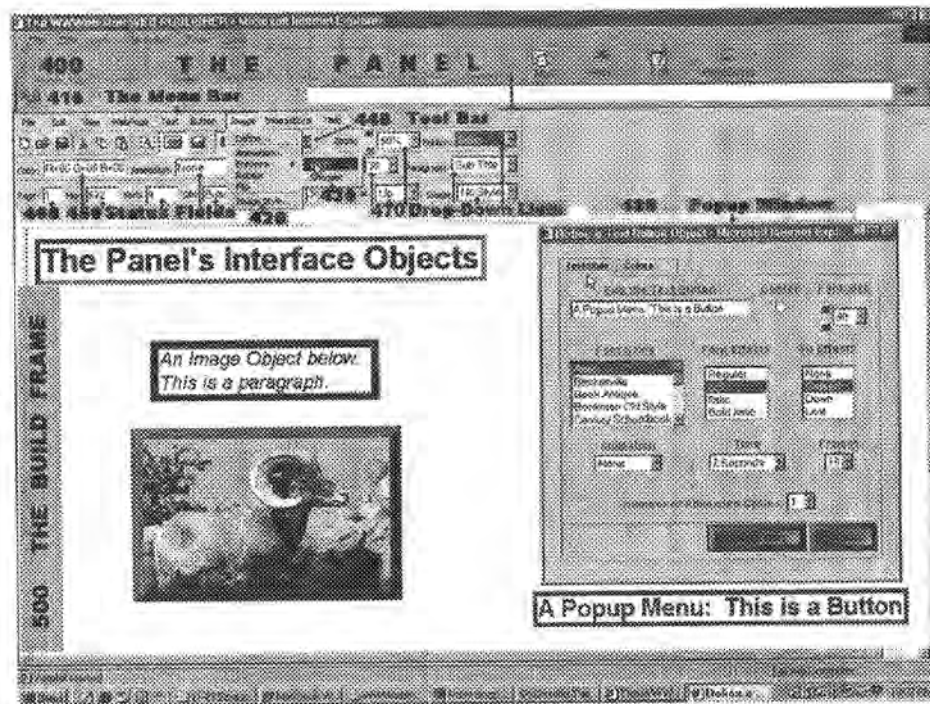
U.S. Patent

Apr. 8, 2003

Sheet 42 of 68

US 6,546,397 B1

Fig. 37





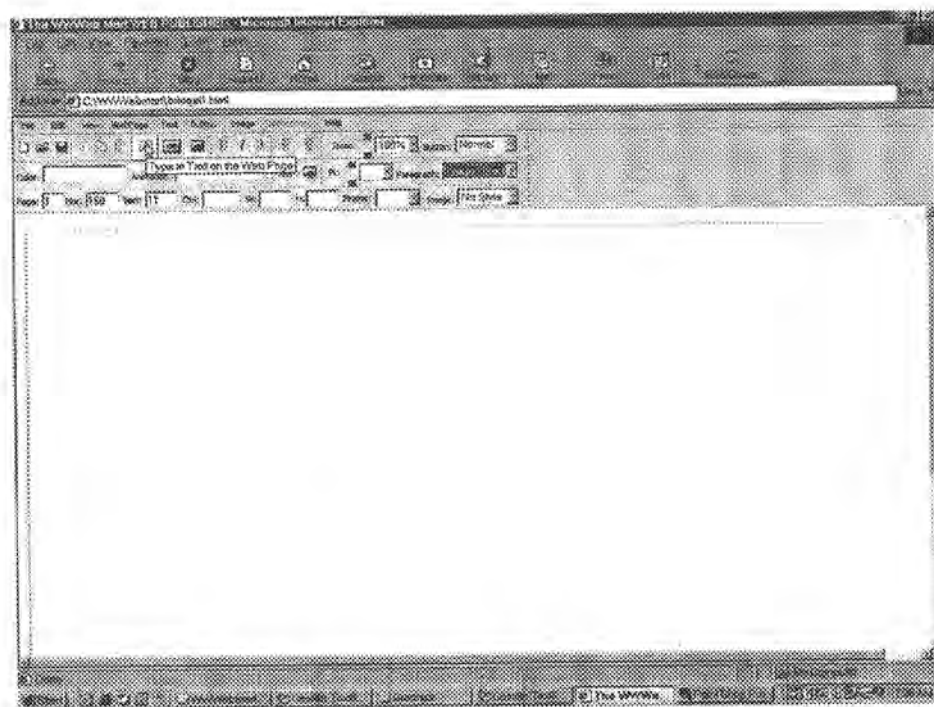


Fig. 38

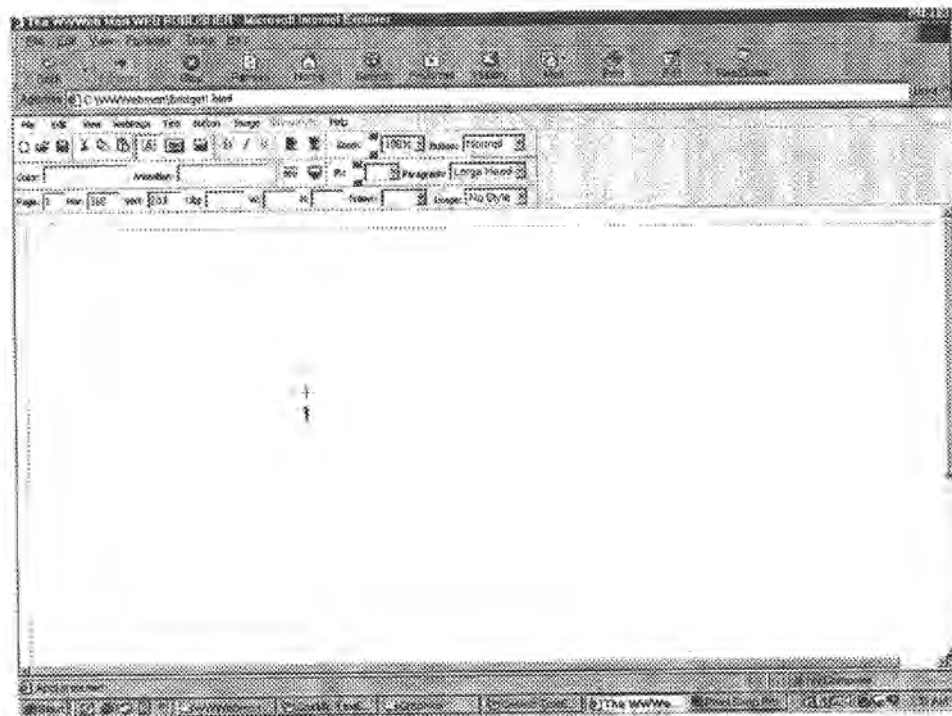
**U.S. Patent**

**Apr. 8, 2003**

**Sheet 44 of 68**

**US 6,546,397 B1**

Fig. 39



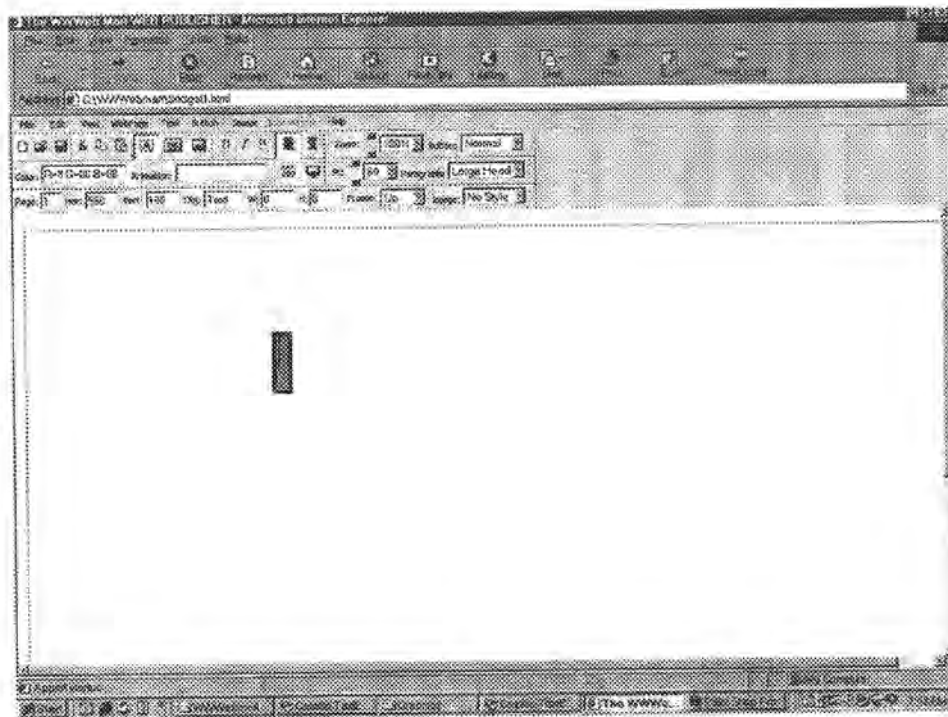
**U.S. Patent**

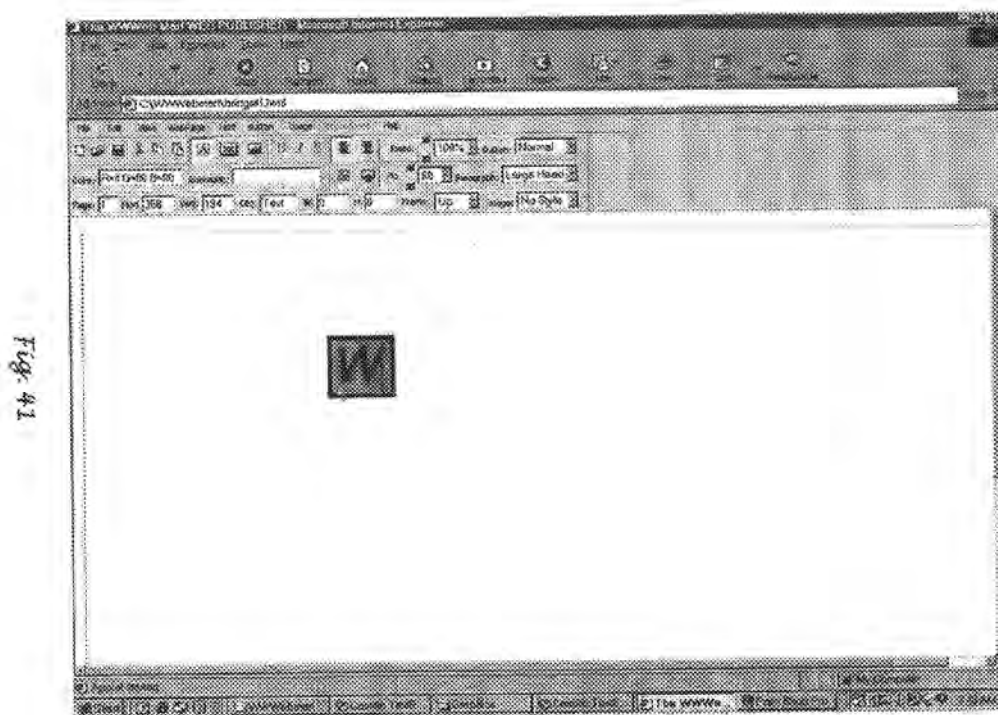
**Apr. 8, 2003**

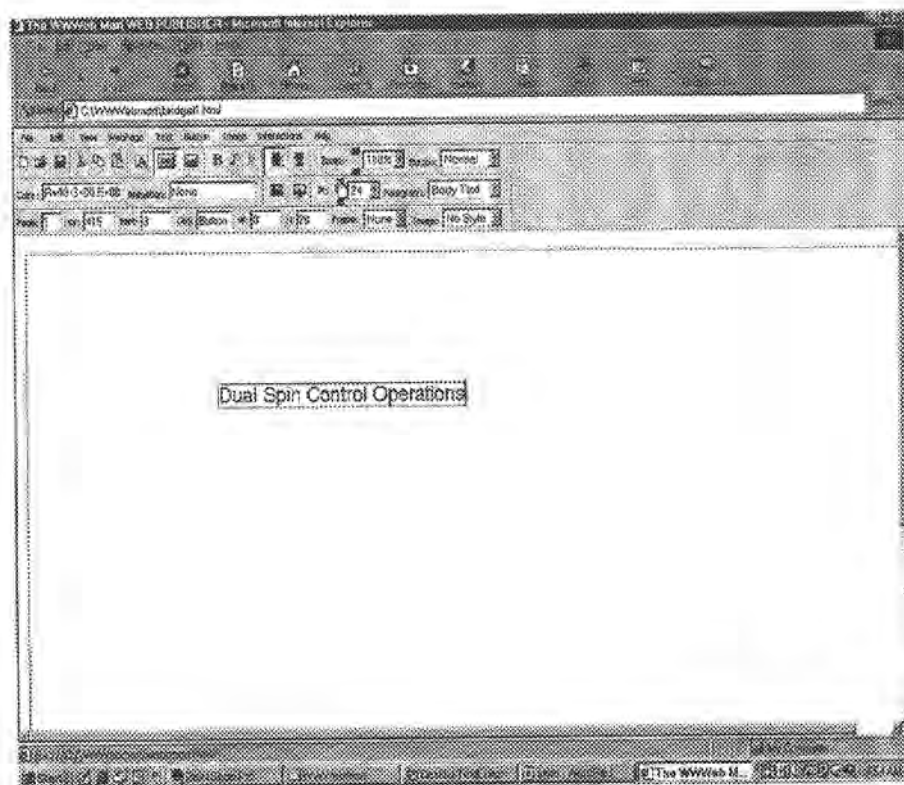
**Sheet 45 of 68**

**US 6,546,397 B1**

*Fig. 40*







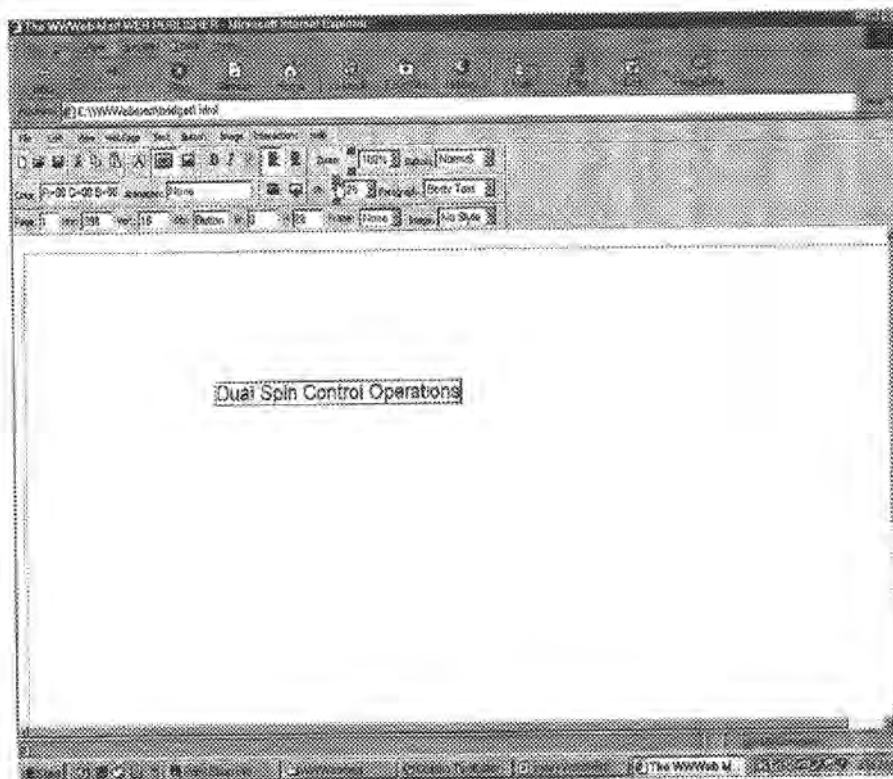
**U.S. Patent**

**Apr. 8, 2003**

**Sheet 48 of 68**

**US 6,546,397 B1**

Fig. 43



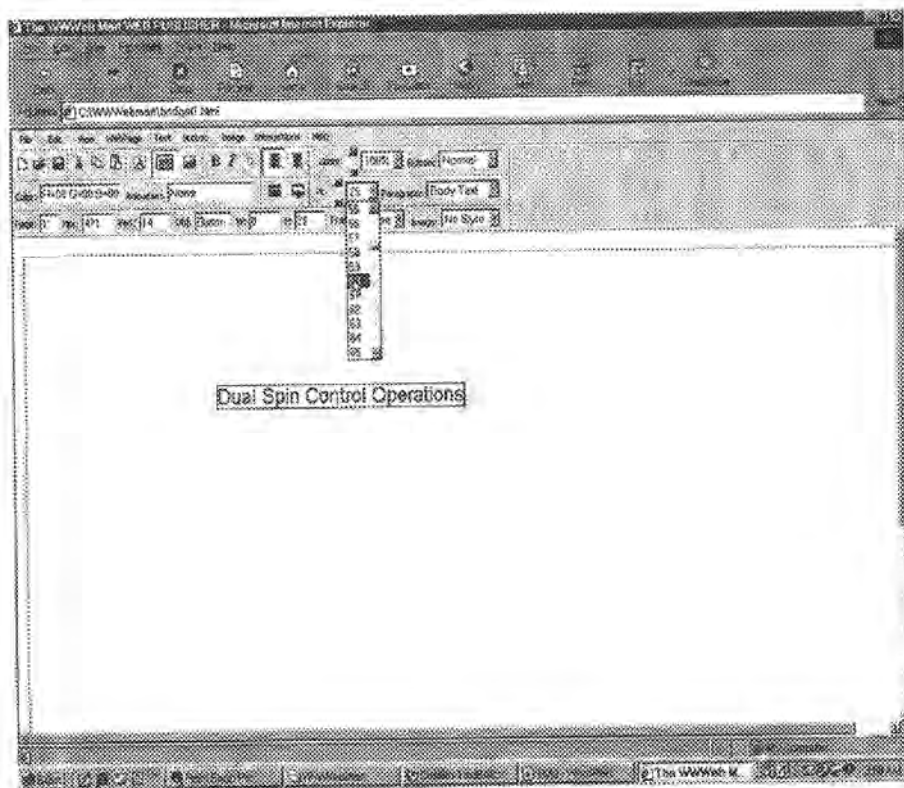
U.S. Patent

Apr. 8, 2003

Sheet 49 of 68

US 6,546,397 B1

Fig. 44



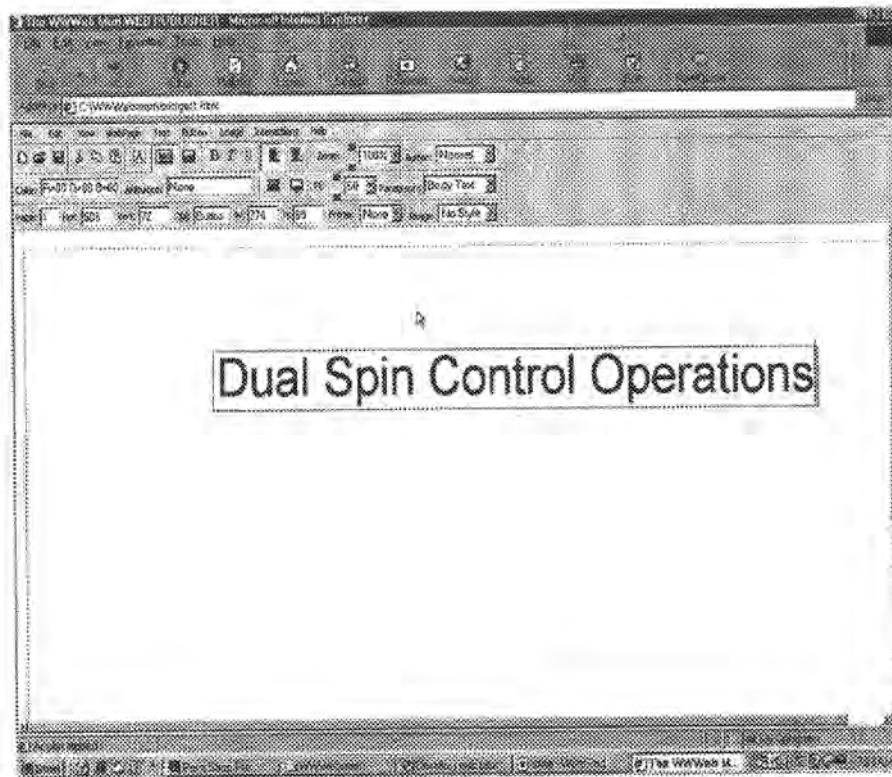
**U.S. Patent**

**Apr. 8, 2003**

**Sheet 50 of 68**

**US 6,546,397 B1**

*Fig. 45*





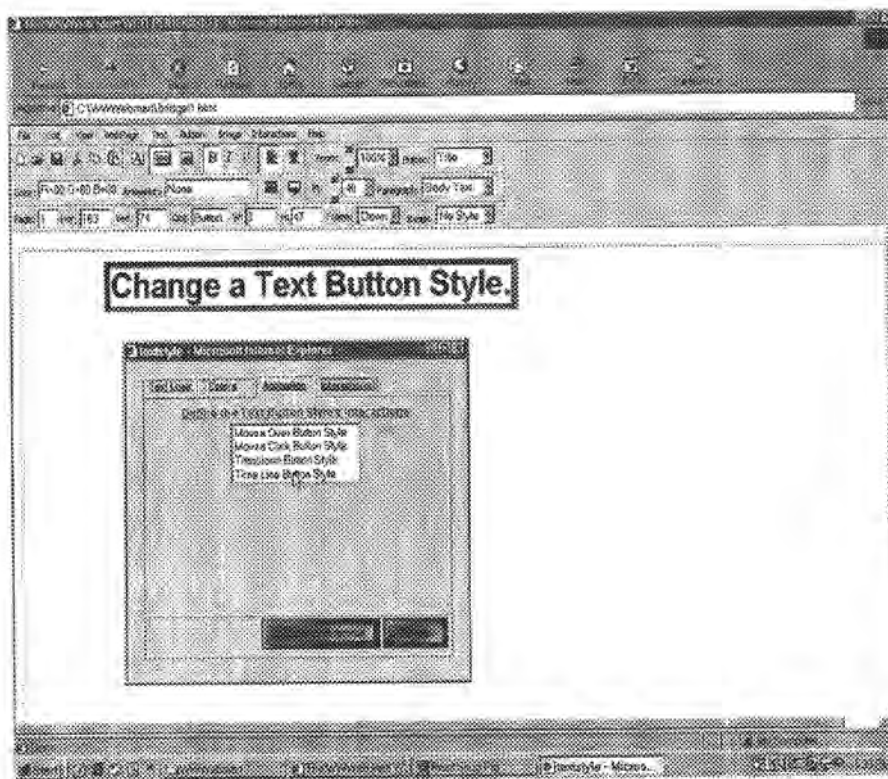


Fig. 46

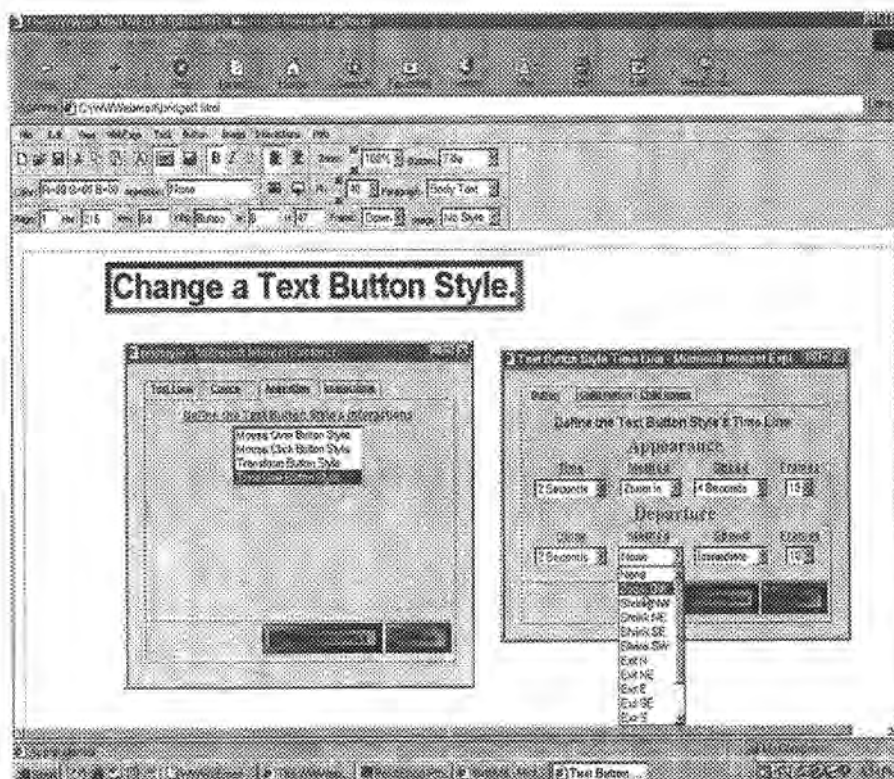
U.S. Patent

Apr. 8, 2003

Sheet 52 of 68

US 6,546,397 B1

Fig. 47



U.S. Patent

Apr. 8, 2003

Sheet 53 of 68

US 6,546,397 B1

Fig. 48

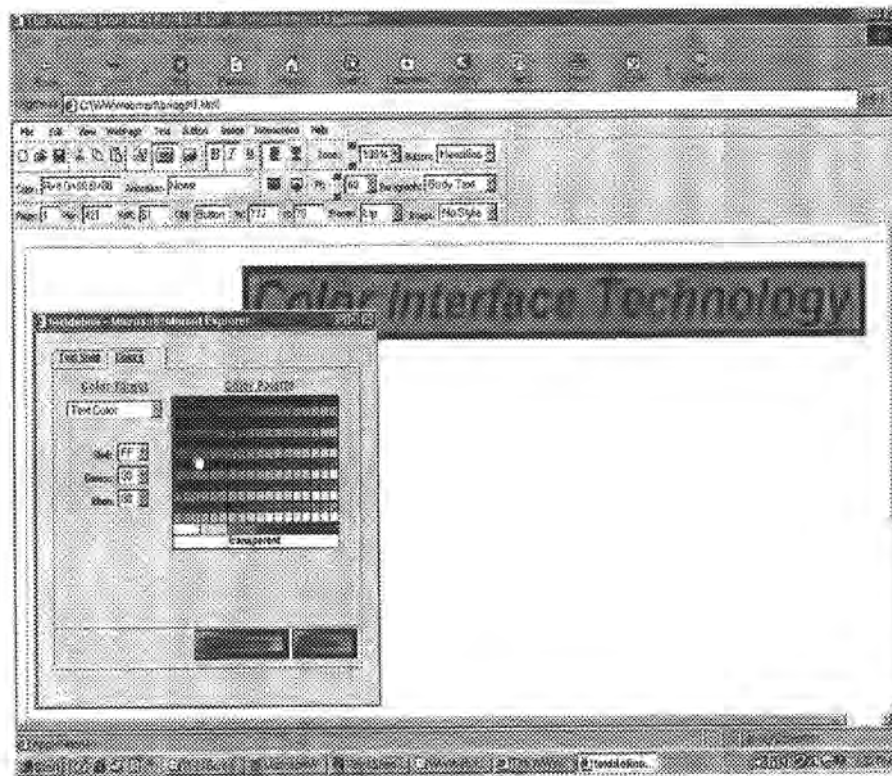
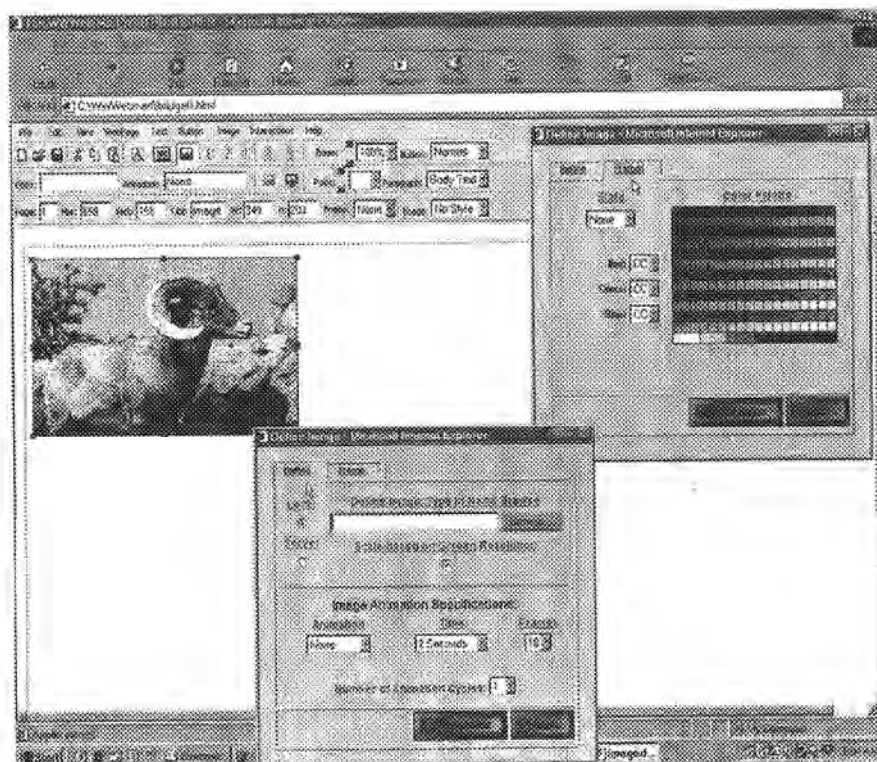


Fig. 49



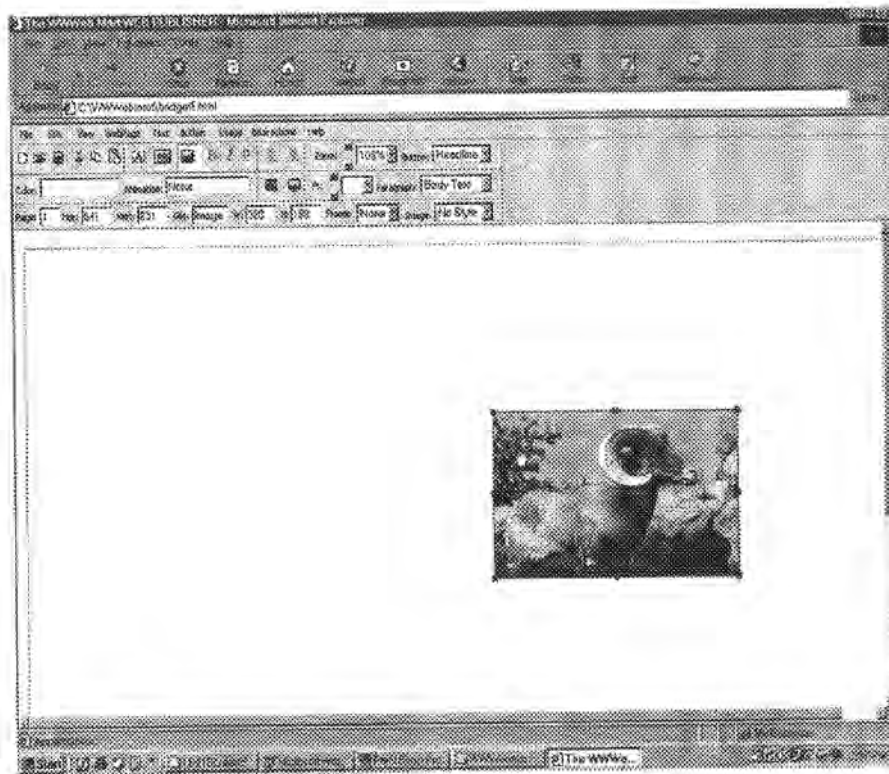
**U.S. Patent**

**Apr. 8, 2003**

**Sheet 55 of 68**

**US 6,546,397 B1**

*Fig. 50*



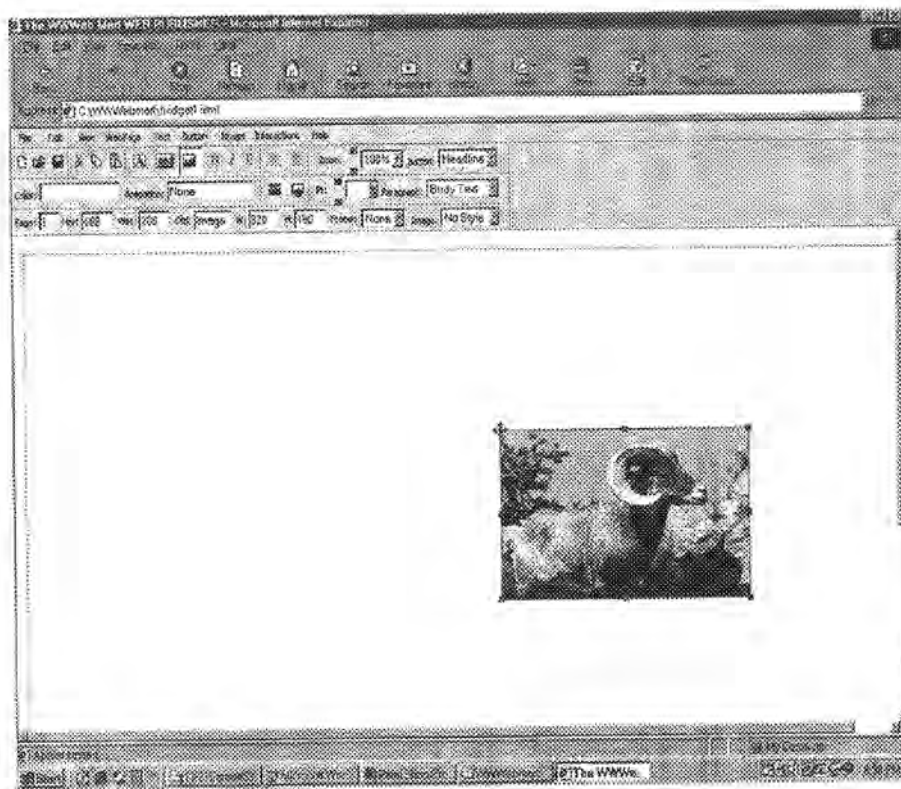
**U.S. Patent**

**Apr. 8, 2003**

**Sheet 56 of 68**

**US 6,546,397 B1**

*Fig. 51*

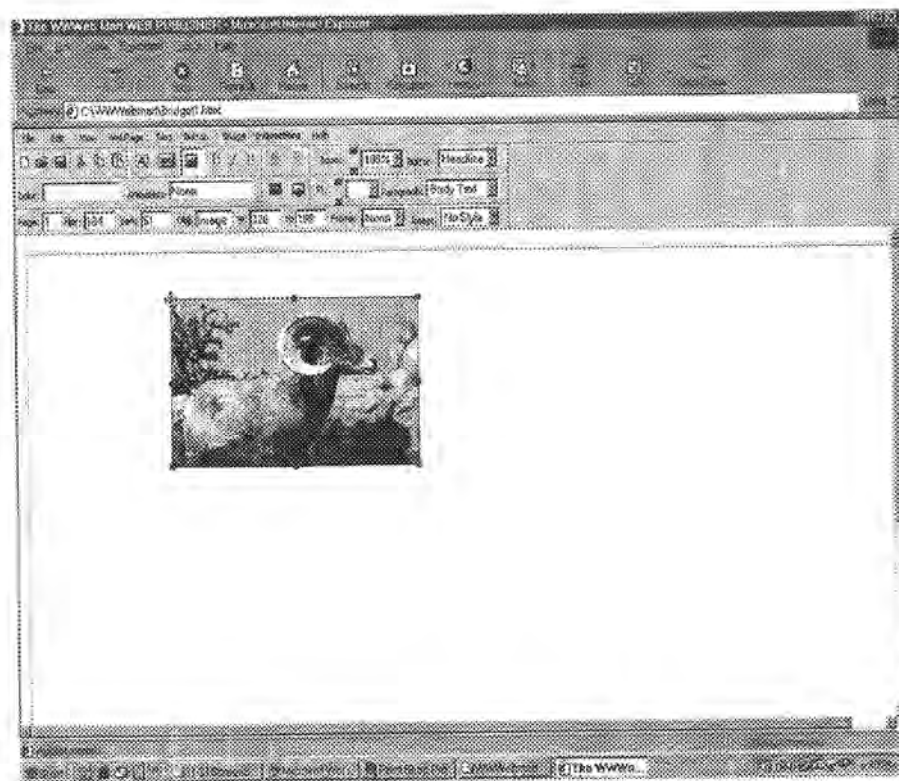


**U.S. Patent**

**Apr. 8, 2003**

**Sheet 57 of 68**

**US 6,546,397 B1**



**Fig. 52**

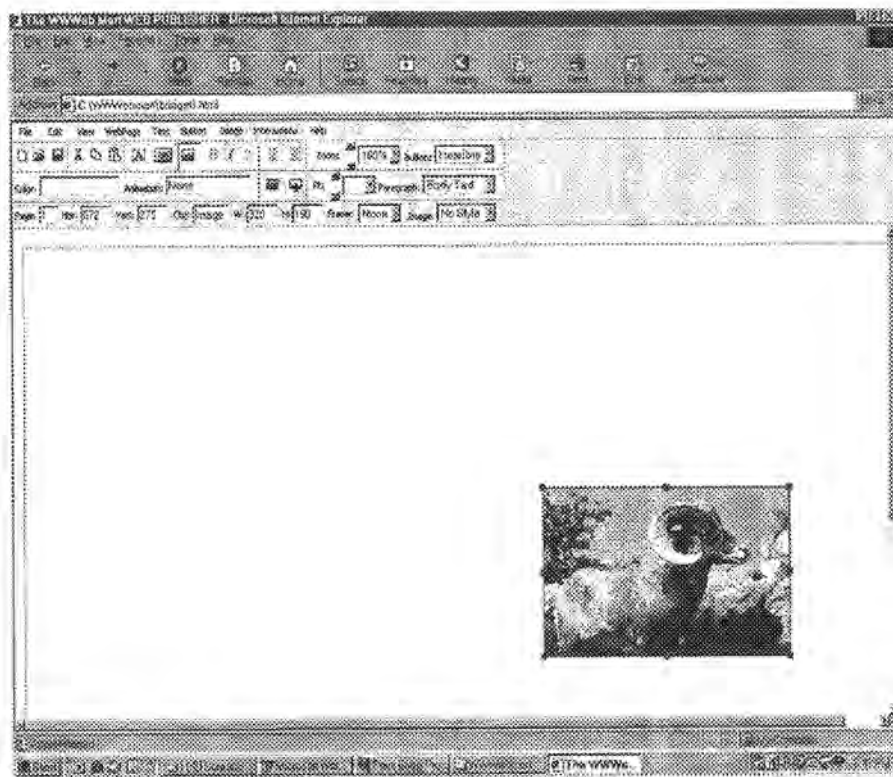
**U.S. Patent**

**Apr. 8, 2003**

**Sheet 58 of 68**

**US 6,546,397 B1**

Fig. 53





**U.S. Patent**

**Apr. 8, 2003**

**Sheet 59 of 68**

**US 6,546,397 B1**

*Fig. 54*



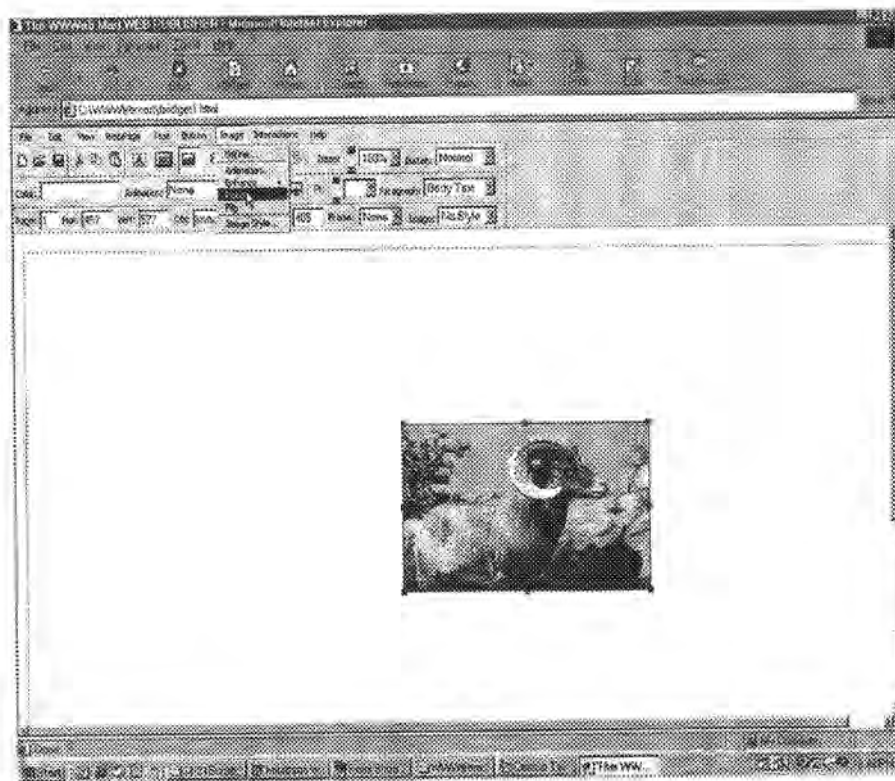
**U.S. Patent**

**Apr. 8, 2003**

**Sheet 60 of 68**

**US 6,546,397 B1**

Fig. 55



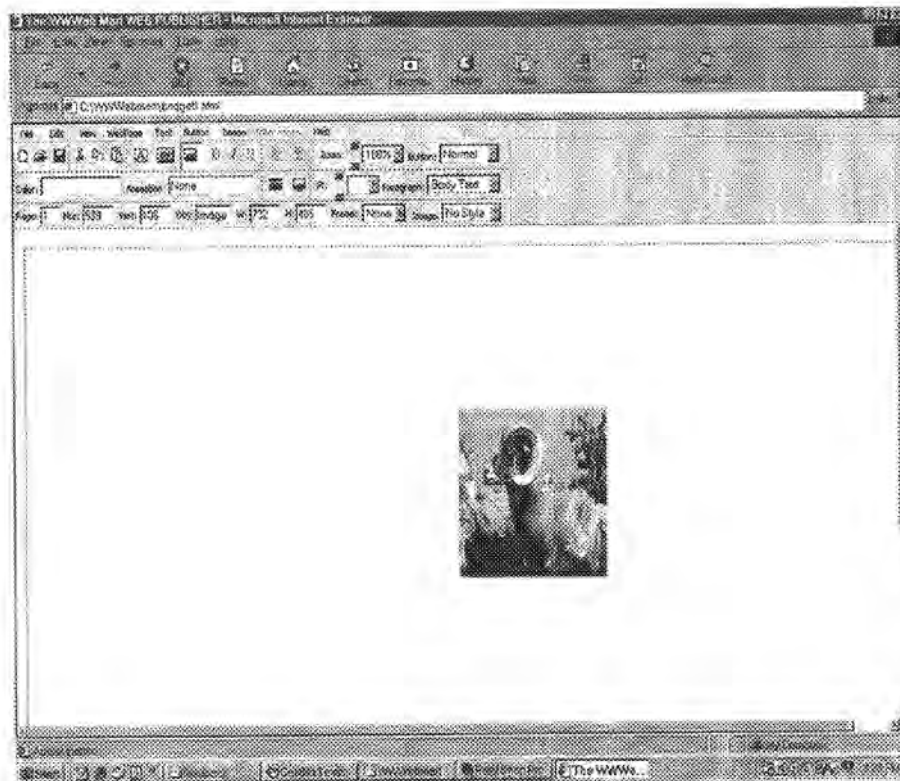
**U.S. Patent**

**Apr. 8, 2003**

**Sheet 61 of 68**

**US 6,546,397 B1**

Fig. 56



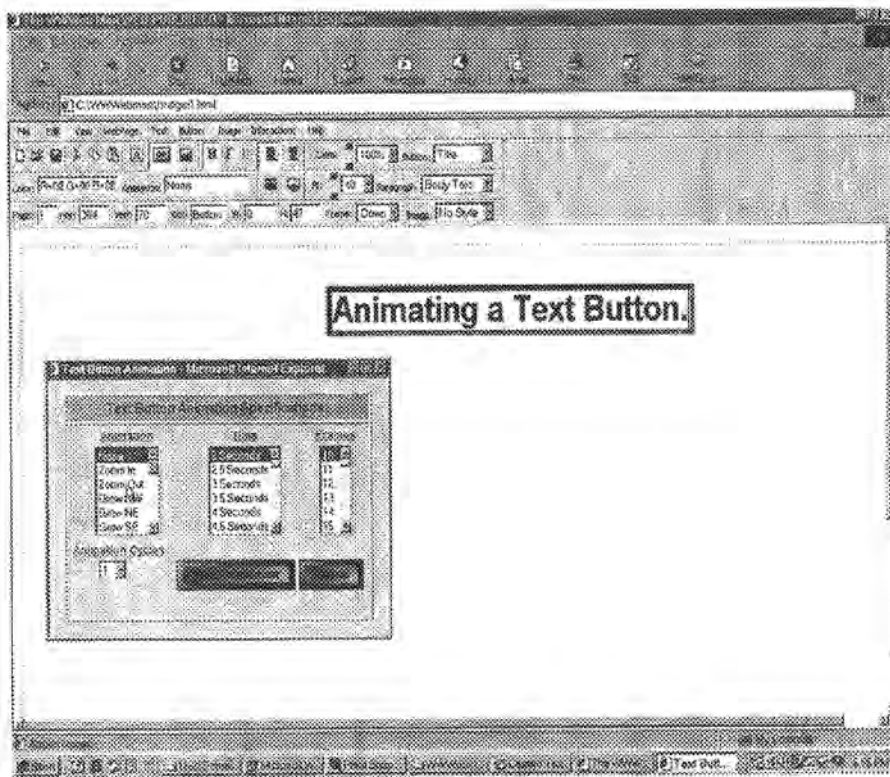
U.S. Patent

Apr. 8, 2003

Sheet 62 of 68

US 6,546,397 B1

Fig. 57



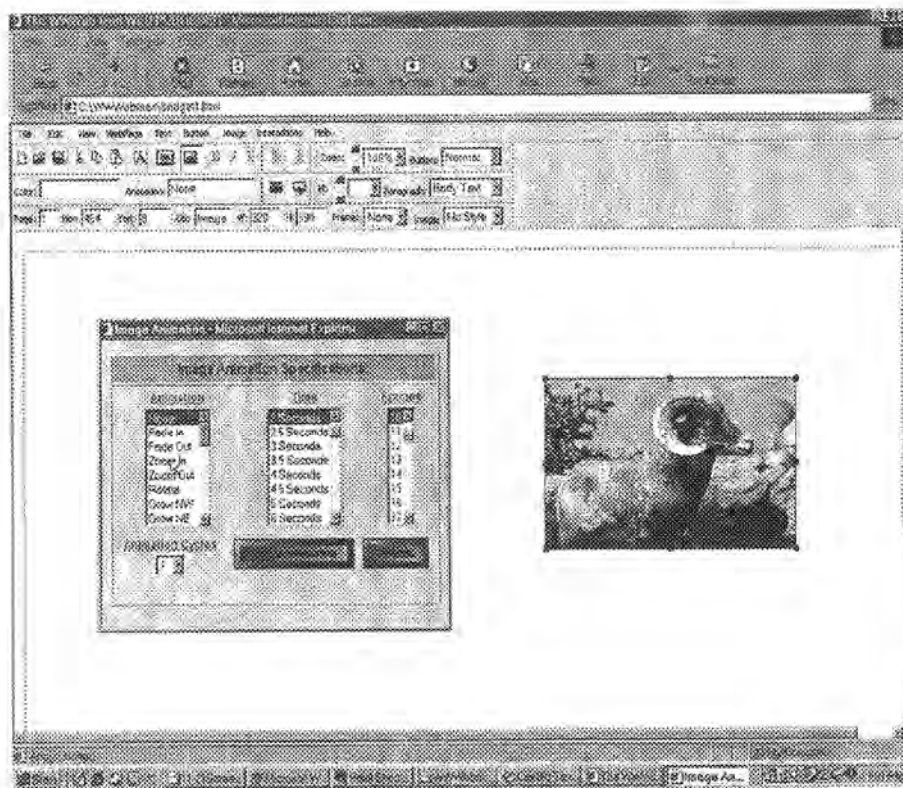
U.S. Patent

Apr. 8, 2003

Sheet 63 of 68

US 6,546,397 B1

Fig. 58



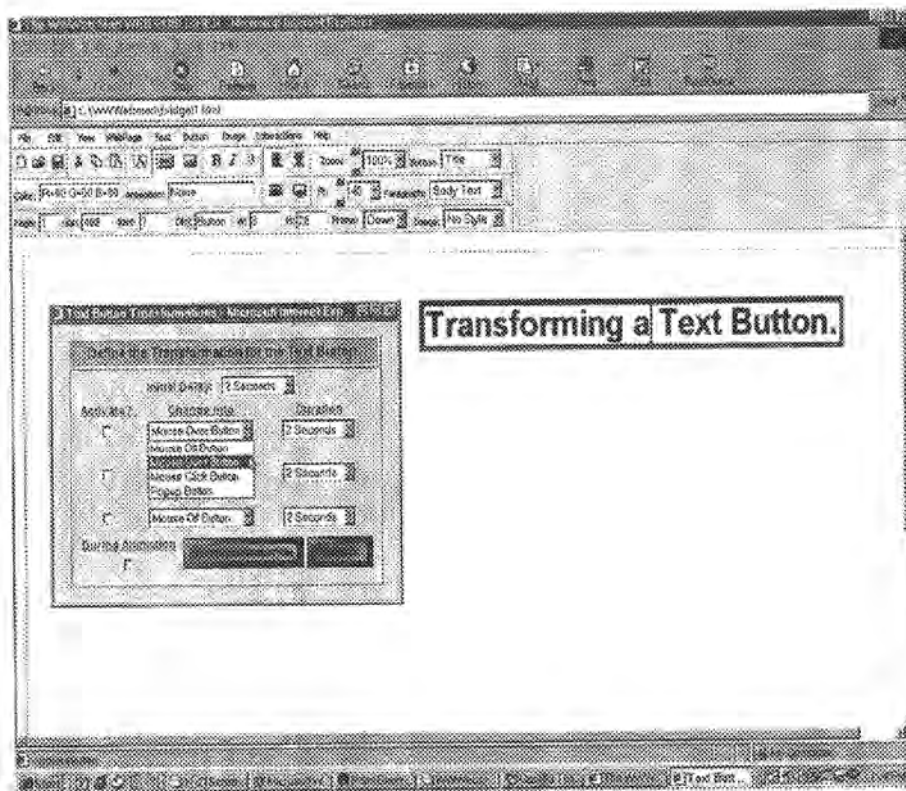
U.S. Patent

Apr. 8, 2003

Sheet 64 of 68

US 6,546,397 B1

Fig. 59



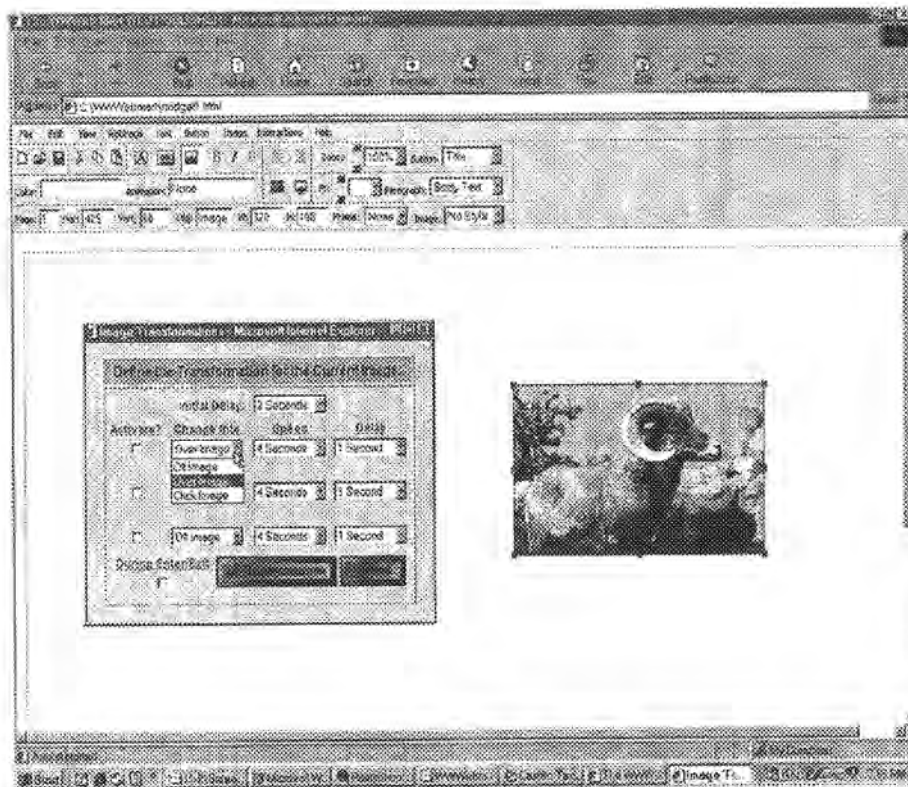
U.S. Patent

Apr. 8, 2003

Sheet 65 of 68

US 6,546,397 B1

Fig. 60



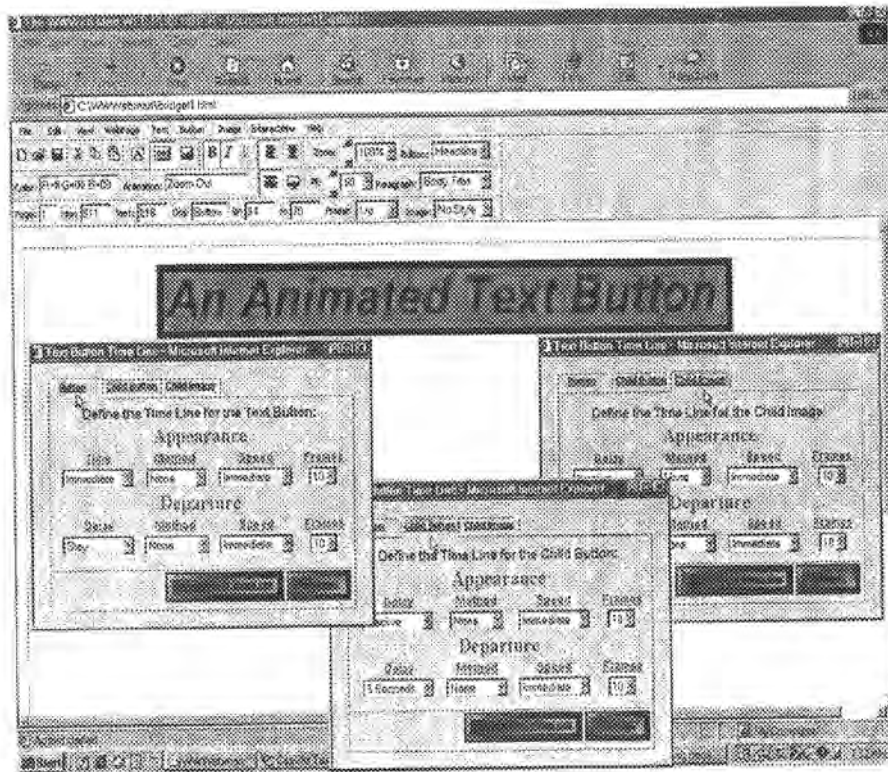
U.S. Patent

Apr. 8, 2003

Sheet 66 of 68

US 6,546,397 B1

Fig. 61





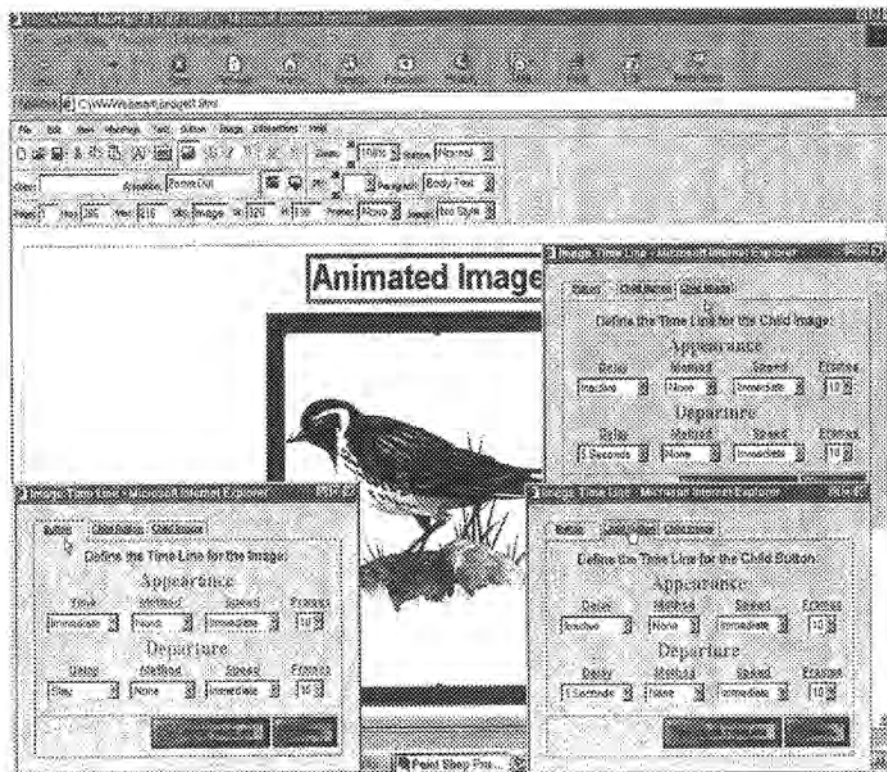
U.S. Patent

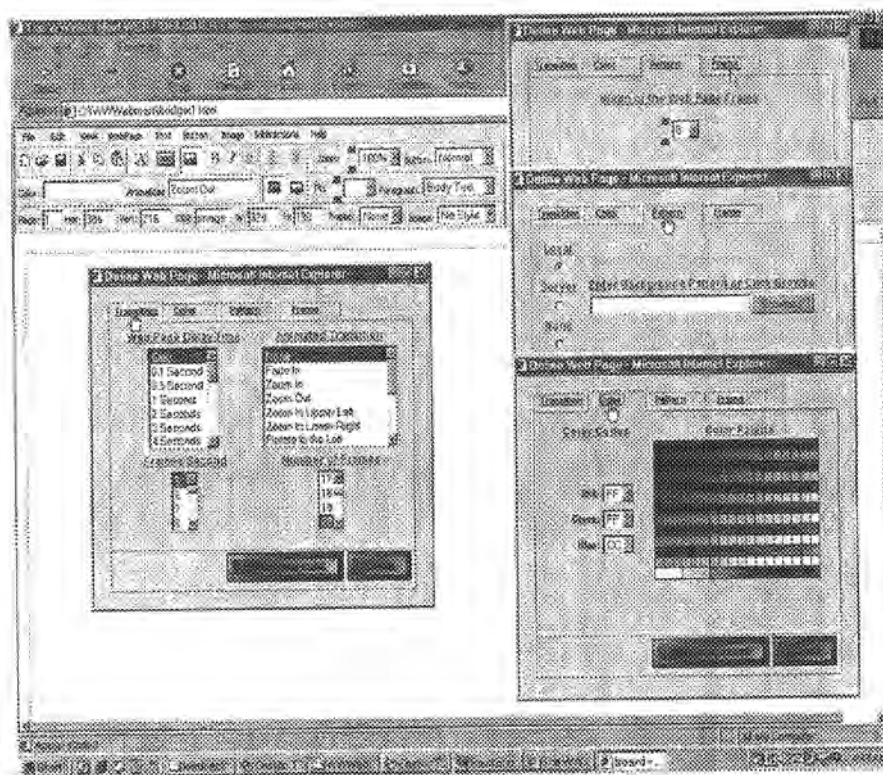
Apr. 8, 2003

Sheet 67 of 68

US 6,546,397 B1

Fig. 62





US 6,546,397 B1

1

**BROWSER BASED WEB SITE GENERATION  
TOOL AND RUN TIME ENGINE****FIELD OF THE INVENTION**

The present application is directed to computing systems, and more particularly to methods and apparatus for building a web site using a browser-based build engine.

**BACKGROUND**

Conventional web site construction tools operate on traditional operating system platforms and generate as output HTML (hyper text mark-up language) and Script Code (e.g., JavaScript). A browser draws a web page associated with the web site by interpreting the HTML and JavaScript Code. However, conventional mark-up and scripting languages include numerous inherent limitations. Conventional mark-up and scripting languages have not been designed for serious multimedia applications. They have almost no file handling ability and very little computational power. In addition, they are remarkably slow and inefficient.

As such it is virtually impossible to write a web publishing application in HTML and JavaScript. All conventional implementations must, and do, utilize a full-featured programming language, such as C++ or Visual Basic. Since the current popular browsers do not support these languages, by necessity, conventional web publishing applications run on platforms other than the World Wide Web (WWW) and its browsers. Therefore, at best, a conventional web publishing application can offer only a crude preview capability of what a real web page will look like.

Although C++ and Visual Basic are very capable languages, the conventional web publishing applications written in these languages are still necessarily limited by the limitations inherent in their form of output, which as described above is typically HTML and scripting code. As such, a conventional web publishing application written in one of these languages suffers from the severe performance problems inherent in these languages.

For example, HTML and JavaScript are incapable of reformatting text and scaling buttons or images dynamically. In addition, most conventional web publishing applications design a web page layout to fit into a 640 pixel wide screen. This means that the ability for higher resolution screens to display more data horizontally is lost. Since capability is wasted on the horizontal plane, unnecessary vertical scrolling may be required. Further, on higher output resolution devices (screens), unsightly extra white space or background may be prevalent.

**SUMMARY**

In one aspect the invention includes a Browser Based build engine that is written entirely in a web based full featured programming language (e.g., JAVA). A Browser Based Interface (the "Interface") between the web designer and the build engine is included. The browser-based interface can be written in the World Wide Web's (WWW) Hypertext Markup Language (HTML) and its Extensions (Dynamic HTML, JavaScript and Cascading Style Sheets). The Interface includes a unique set of communication techniques. One technique allows for effective two-way communications between a JAVA engine and JavaScript. Another technique allows for communications between a JAVA applet object inside a JavaScript window, with the JAVA engine, which permits the implementation of advanced intelligent interface objects, such as a "slider" or a "dial".

2

In one aspect the invention includes a screen resolution sensing mechanism that causes a build engine (i.e. build tools) to adopt its interface to the web designer's screen resolution.

5 In one aspect, the invention includes a multi-dimensional array structured database, that, in addition to storing the numeric and string data found in conventional databases, also stores multi-dimensional arrays of various multimedia objects. They include colors, fonts, images, audio clips, video clips, text areas, URLs and thread objects. The invention includes a run time generation procedure that creates a compressed web site specific customized run time engine program file, with its associated database and a build engine generated HTML shell file.

10 The invention can include web page scaling technology, so that when the web site/web page is accessed on the WWW, the web pages and all the objects within them can be scaled to the user's screen resolution and to the then current browser window size.

15 In one aspect, the invention includes a proprietary multi-level program animation model (threads) that responds to multiple user interactions and time sensitive operations simultaneously.

20 In one aspect, the invention includes a mechanism for the dynamic resizing of the build engine's web page size during various editing operations.

25 In one aspect, the invention includes techniques for creating browser based interface objects that visually and behaviorally are identical to those of the MS Window's standard.

30 Aspects of the invention can include one or more of the following features. A browser based build engine is provided that includes a browser based interface. The entire web site build process is WYSIWYG (what you see is what you get), with the web designer working directly on and with the final web page. The data produced by the build engine is processed and ultimately placed into a multi-dimensional array structured database, and stored in an external file. A run time generation procedure creates a compressed program customized run time engine file, with an associated database and a build engine generated HTML Shell File.

35 When the web site/web page is accessed on the WWW, web page scaling technology can be accessed to generate web pages that are scaled to the user's screen resolution. A technique is provided so that an applet's size (height and width) can be set in real time under the control of either the interface or the build engine. At the same time a multi-level program animation model (threads) is activated for user interactions and time sensitive operations.

40 The browser based interface technologies create a set of interface objects with a look and feel that is identical to that of MS Windows, yet includes technologies that equal or occasionally surpass those of high end word processors, desk top publishers, and image processing software programs, particularly in the areas of interaction, animation, and timeline technologies. The run time engine includes multimedia capabilities often rivaling the digital processing capabilities seen on television and in the movies.

45 Because of the implementation of a variety of performance and file reduction techniques, the entire run time environment can range from as low as 12K, and no larger than 50K. This depends upon the features selected by the web designer. Although the compressed image, audio, and/or video files must also be downloaded, with a reasonable web site design, web pages should load quickly. The initial run time environment is no larger than 25K, thus the initial

US 6,546,397 B1

3

web page should generally load in less than 2 seconds, and subsequent web pages in less than 1 second with a 56K modem, even with numerous image files.

The present invention provides a real time, dynamic linkage between JAVA and HTML including two-way communications, in real time, between JAVA and JavaScript.

#### BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing aspects and many of the attendant advantages of the invention will become more readily appreciated through the following drawings and their associated screen shots, referred to throughout the detailed description, wherein:

FIG. 1 is a flow chart depicting a prior art conceptual overview of how a user and a web browser interface.

FIG. 2 is flow chart depicting a conceptual overview of how a user interfaces with a web browser when implementing the present invention to construct a web site.

FIG. 3a is a schematic diagram showing the main components of a build tool in accordance with one implementation of the present invention.

FIG. 3b is a process flow diagram showing a build process in accordance with one implementation of the present invention.

FIG. 4a is schematic diagram showing the main components of a run generation tool in accordance with one implementation of the present invention.

FIG. 4b is process flow diagram showing a run time process in accordance with one implementation of the present invention.

FIG. 5 is a flow chart, with its attendant screen shot shown in FIG. 37, that depicts a detailed view of a build time initialization procedure in accordance with one implementation of the present invention.

FIG. 6 is a flow chart, with its attendant screenshots shown in FIGS. 38-48, that depicts a detailed view of the build time supported user input techniques and techniques for communication of data and status between the build engine and the interface in accordance with one implementation of the present invention.

FIG. 7a is a flow chart that shows an overview of the build time techniques for implementation of pop-up windows (usually called "dialog boxes" in MS Windows), the panel interface, and interface for color selection.

FIG. 7b is a flow chart, with its attendant screenshots shown in FIGS. 37-38, that shows a detailed view of the build time techniques for implementation of panel interface objects, including the menu bar, menus and sub-menus, the tool bars, status fields, interactive fields, and interactive pull down lists, in accordance with one implementation of the present invention.

FIG. 7c is a flow chart, with its attendant screenshots shown in FIG. 37 and FIG. 63, that shows a detailed view of the build time techniques for implementation of tabbed pop-up windows (also called "dialog boxes" in MS Windows).

FIG. 8 is a flow chart that shows a detailed view of the build time techniques for updating the internal databases and the setting of feature flags for run time optimization purposes.

FIG. 9 is a flow chart, with its attendant screenshot shown in FIG. 37, that shows a detailed view of the build time polling methods used to facilitate communication from the JAVA build engine to the interface.

4

FIG. 10 is a flow chart that shows a detailed view of the build time techniques for analyzing user input for error checking and data integrity.

FIG. 11 is a flow chart, with its attendant screenshot shown in FIGS. 38-41, that shows a detailed view of the build time methods for direct text entry at an arbitrary cursor position and text editor implementation methods.

FIG. 12 is a flow chart, with its attendant screenshot shown in FIGS. 49-56, that shows a detailed view of the build time techniques for reading external image files, creating them on a web page, and then manipulating them through either direct mouse interaction or through the interface's panel/windows.

FIG. 13 is a flow chart that shows a detailed view of the build time implementation of text, button and image styles in accordance with one implementation of the present invention.

FIG. 14 is a flow chart that shows a detailed view of the video and audio processing in accordance with one implementation of the present invention.

FIG. 15 is a flow chart that shows a detailed view of the frame, table, forms, and draw objects processing and technology in accordance with one implementation of the present invention.

FIG. 16 is a flow chart that shows a detailed view of the build time methods for supporting various user interactions at run time.

FIG. 17 is a flow chart, with its attendant screenshots shown in FIGS. 57-58, that shows a detailed view of the build time methods for text button and image object animation.

FIG. 18 is a flow chart, with its attendant screenshots shown in FIGS. 59-60, that shows a detailed view of the build time methods for text button and image transformations.

FIG. 19 is a flow chart, with its attendant screenshots shown in FIGS. 61-62, that shows a detailed view of the build time methods for text button and image time lines.

FIG. 20 is a flow chart with its attendant screenshot shown in FIG. 63, that shows a detailed view of the build time web page transition animations and time lines.

FIG. 21a is a flow chart that shows a detailed view of file operations performed in accordance with one implementation of the present invention.

FIG. 21b is a flow chart that shows a detailed view of the view operations performed in accordance with one implementation of the present invention.

FIG. 22 is a flow chart that shows a detailed view of a dynamic web resizing process that is activated by the "Open" and "Web Site" commands under the "File" menu and the "Zoom" command under the "View" menu.

FIG. 23 is a screen shot showing a file selection window operation in accordance with one implementation of the present invention.

FIG. 24 is a flow chart showing a detailed view of an external database in accordance with one implementation of the present invention and also shows the security and optimization techniques that can be employed.

FIG. 25 is a flow chart showing a detailed view of a method for creating a customized and optimized run time engine in accordance with one implementation of the present invention.

FIG. 26 is a flow chart showing a detailed view of the methods for creating an HTML shell file in accordance with one implementation of the present invention.



US 6,546,397 B1

5

FIG. 27 is a flow chart showing a detailed view of the methods for creating compressed CAB and JAR files in accordance with one implementation of the present invention.

FIG. 28 is a flow chart showing a detailed view of the technology for dynamic web page size creation in accordance with one implementation of the present invention.

FIG. 29 is a flow chart showing a detailed view of the methods for reading the multimedia database and generating the necessary objects in accordance with one implementation of the present invention.

FIG. 30 is a flow chart showing a detailed view of the methods for dynamically scaling the web page object(s) to different screen resolutions and window sizes in accordance with one implementation of the present invention.

FIG. 31 is a flow chart showing a detailed view of the methods for executing a multi-level web page and object thread architecture in accordance with one implementation of the present invention.

FIG. 32 is a schematic diagram that shows a detailed view of the web page transition animation architecture in accordance with one implementation of the present invention.

FIG. 33 is a schematic diagram that shows a detailed view of the parent object time line architecture in accordance with one implementation of the present invention.

FIG. 34 is a schematic diagram that shows a detailed view of the child object time line architecture in accordance with one implementation of the present invention.

FIG. 35 completes the flow chart begun at FIG. 31.

FIG. 36 is a flow chart showing a detailed view of the methods for responding to user interactions in accordance with one implementation of the present invention.

FIGS. 37-63 are screen shots of the user interface presented by the build process in accordance with one implementation of the present invention.

#### DETAILED DESCRIPTION

Referring to FIG. 1, in a prior art process for creating and displaying a web site, the user either directly writes HTML and Script Code providing user input at 1 or operates a related prior art product at 2, which generates the HTML and Script Code at 3. A separate file, with its attendant HTML and Script Code is uploaded for each separate web page in the web site at 4, which is then interpreted by a browser when accessed at 5.

FIG. 2 shows a process for creating and displaying a web site in accordance with one aspect of the invention in which, a user operates a build tool at 6, working directly with one or more of the final web pages in a full WYSIWYG mode. The build tool accepts the user input and creates a multi-dimensional embedded multimedia object database at 7. A run time generation process is then invoked to create the necessary run time files at 8 (including HTML shell, CAB/JAR files and a customized runtime engine) which are then loaded to a user's web site at 9. The web page(s), when viewed by a web surfer, are activated by the browser calling the customized run time engine at 10. The run time engine then begins to read the database and down load image, audio and video files, while simultaneously drawing the first web page for viewing or user interaction at 11.

FIG. 3a shows a build tool 350 at the detailed component level. The build tool includes a build engine 352, interface 354, screen sensing mechanism 356, multi-dimensional array structured database 358, interface's database 360, web

6

page scaling engine 364, time line engine 366 and installation Program 368. The operation and use of each of these components is described in greater detail below.

FIG. 3b is a flow of the build process executed by the build tool to create a web page/web site. Referring to FIGS. 3a and 3b, the process begins with an initialization (12) and continues through to a point where a web site has been defined and stored in the build engine's internal database (29).

The build tool 350 includes plural individual tools that are created and initialized at (12). The processes for creating and initializing build tools are described in greater detail below in association with FIG. 5. After the build tools are created and initialized at 12, the build tool 350 interacts with the user, receiving user commands (actions), for example, to build a web site. The build tool 350 processes user responses and communicates the same and status information to both the build engine 352 and interface 354 at 13. The processes for interacting with the user are described in greater detail below in association with FIG. 6.

In one implementation, the interface includes a panel (and its objects, including a menu bar, menus and sub-menus, tool bars, status fields, interactive fields and interactive pull down lists), pop-up windows (called "dialog boxes" in MS Windows), color and alert message interface technologies, built with HTML, Dynamic HTML (DHTML), JavaScript, and Cascading Style Sheets (CSS). Interface 354 responds to the user input and may display a pop-up window, update the interface objects, or display alert messages, as shown at 15. The operation of the interface 354 is described in greater detail below in association with FIG. 7a, FIG. 7b and FIG. 7c.

As the build engine 352 receives data and status information, it updates an internal database (part of multi-dimensional array structured database 358) and sets feature flags at 14. The processes for updating the internal database and setting flags are described in greater detail below in association with FIG. 8. To enable effective two-way communication between the interface and the build engine, polling technology is included as shown at 16. The details of the polling process are described in greater detail below in association with FIG. 9.

Whenever user input is received, the build tool 350 analyzes the input including error checking at 17. In one implementation, the input is analyzed and then processed by object type (class). The process for analyzing input to determine type is described in greater detail below in association with FIG. 10. In one implementation, the number of different object processing technology classes are four, and include direct text entry (18), image processing (19), video or audio files and links (21) and frames, tables, forms and draw objects (22). The build tool 350 processes the user input based on class. The processes invoked for direct text entry are described in greater detail below in association with FIG. 11. The processes invoked for image processing is described in greater detail below in association with FIG. 12. The processes invoked by the text button, paragraph, and image style technologies are described in greater detail below in association with FIG. 13. The processes invoked for processing audio and video files and channels are described in greater detail below in association with FIG. 14. The processes invoked for processing frames, tables, forms and draw objects are described in greater detail below in association with FIG. 15. When an image, text button or paragraph object is to be inserted in the web page, the current style that is selected in the panel defines the initial settings used when creating the object in the web

US 6,546,397 B1

7

page. As such, button, image and paragraph style setting and technology will be invoked at 20 depending on the user input. The processes invoked by the paragraph style setting and technology is described in greater detail below in association with FIG. 13.

After the input is processed as described above, a check is made to determine if one or more animation or transformation (interaction) techniques are to be invoked at 23. The run time engine provided in accordance with the teachings of the present invention support various user interactions, including support for numerous animation and transformation techniques, and both web page and object time lines. Depending on the user selections, one or more technologies may be invoked. In the implementation shown, the build tool 350 is configured to check to determine if the input data is related to plural technologies including: user interaction technology (24), animation technology (25), transformation technology (26), object time line technology (27) and web page transition animation technology (28). The processes invoked for user interaction technology are described in greater detail below in association with FIG. 16. The processes invoked for animation technologies are described in greater detail below in association with FIG. 17. The processes invoked for transformation technologies are described in greater detail below in association with FIG. 18. The processes invoked for object timeline technologies are described in greater detail below in association with FIG. 19. The processes invoked for web page transition animation technologies are described in greater detail below in association with FIG. 20.

After the build tool 350 has processed the user input, one or more file operations can be invoked at 29a. In one implementation, the file operations are "save", "save as", "new", "close", "open", "apply" and "web site". If "open" or "web site" are selected, the build tool 350 initiates the dynamic web page resizing process at 29c (See FIG. 22). If "save" or "save as" are selected, the build tool 350 initiates a run generation process (See FIG. 4 and FIG. 24). File operations "close", "open", and "new" can also initiate the run generation process, based on the state of the build process and user action.

At any time during the processing of user input, one or more view operations can be invoked at 29b. In one implementation, the view operations supported are "normal", "preview", "play", and "zoom" (at various zoom percentages). If any of the "zoom" levels are selected, the build tool initiates the dynamic web page resizing process at 29c (See FIG. 22). If the "preview" or "play" view operations are selected they will initiate the run time process (See FIGS. 28 through 36). FIG. 4a shows a run generation and runtime tool 370 at the detailed component level. The run generation and runtime tool 370 includes a run generation procedure 371, web scaling engine 372, a database 374 and a (web) page size generation engine 376 and run time engine 377 including a runtime user interaction engine 378, a runtime timeline engine 380 and a runtime drawing, animation, audio, and video engine 382. In one implementation, run time engine 377 includes plural engines, each of which may in themselves include plural engines.

FIG. 4b shows the run processes including methods for creating the run time files, including the external database, the web site specific customized run time engine, the HTML shell file, and the compressed CAB/JAR file. The run processes also include methods for scaling each web page to the web surfer's then current screen resolution and web browser window size. After a web page has been scaled, a

8

run time engine executes a multi-level thread technology, which presents to the viewer web pages that can operate under time lines that may include animated transitions. Associated with the web page time lines can be object time lines that may define entrance, main and exit animations, transformations, and synchronized time lines for child objects. Each object can have multiple object states, responsive to various user interactions, which can result in numerous types of visual and audio responses and actions.

Referring now to FIGS. 4a and 4b, a run generation process 360 begins by invoking the run generation procedure 377. The run generation procedure 371 begins by creating the external database (part of database 374) at 30. The external database may include references to image, video and audio files, and video and audio channels. The process for creating the external database is described in greater detail below in association with FIG. 24. A customized and optimized run time engine (run time engine 377) is created at 31. The customized and optimized run time engine (run time engine 377) generates the web pages for the web site and is activated from the user's server. The process for creating the run time engine 377 is described in greater detail below in association with FIG. 25. The HTML shell file is created at 32, and then the CAB and JAR files are created at 33a. The HTML shell file includes JavaScript Code to activate and interrogate the page size generation engine 376, and to activate the entire runtime engine. The CAB and JAR files both include the runtime engine and database in compressed executable form. The CAB file(s) will be activated by the HTML shell file if it senses the browser as being Microsoft Explorer, otherwise it will activate the JAR file(s). The processes for creating the HTML shell file and the CAB and JAR files are described in greater detail below in association with FIG. 26 and FIG. 27, respectively. The run generation process portion of the run processes is completed as the HTML shell file and the CAB and JAR files are uploaded to the user's web site at 33b.

After the upload, the run time process 365 portion begins with the run time engine 377 invoking a web page size generation technology (engine) 376 at 34. The web page size generation technology can be used to determine the screen resolution and the current browser window size. The process for invoking and initializing the web page size generation technology is described in greater detail below in association with FIG. 28. The external database is read and the necessary objects generated at 35 from their stored external references. These objects include image, audio, and video objects. The processes for generating the necessary objects are described in greater detail below in association with FIG. 29. A web page generation and scaling technology (web page scaling engine 372) is then invoked at 36. The web page scaling engine 372 can be used to reformat and scale objects that had been placed in a web page during the build process. The processes employed by the web page generation and scaling technology are described in greater detail below in association with FIG. 30. The run time engine then, as necessary, executes a multilevel web page and object thread technology at 37 while the runtime user interaction portion 378 of run time engine 371 responds to user interactions at 38. The processes invoked by the multilevel web page and object thread technology are described in greater detail below in association with FIGS. 31-35. The processes invoked by the run time engine to respond to user interactions are described in greater detail below in association with FIG. 36.

Detailed Build Processes

Referring now to FIGS. 3a and 5 through FIG. 22 the build tool 350 and its associated build process are described.

US 6,546,397 B1

9

Referring first to FIGS. 3*a* and 5, initialization methods are shown. At 39 the build tools are created as part of the execution of the installation program 368. They can include:

- 1: Initial build tool HTML/JavaScript file (IBTF)
- 2: An initialization engine (IE).
- 3: A build engine.
- 4: The build engine parent HTML frame file. (PFF).
- 5: A "Control Panel and Status Line" HTML/JavaScript File ("panel") for:
  - Controlling the JavaScript database.
  - Calling and initializing all pop-up windows.
  - Reading all pop-up window values, and updating a JavaScript database
  - Calling the build engine and passing all necessary data and status information.
  - Polling the build engine for two-way JAVA/JavaScript communication.
  - Displaying and updating the status of its interface objects.
  - Issuing alert messages.
  - Processing direct user interactions with the panel's interface objects.
- 6: Numerous HTML/JavaScript files, one for each pop-up window.
- 7: JAVA applets, embedded in HTML/JavaScript pop-up window files.
- 8: A build engine HTML definition file that is created and modified dynamically.

The initialization and build engines can be placed in a JAVA wrapper so that JavaScript code may receive and process return values from JAVA methods. The initialization and build engines are also created in a "Signed" CAB file, and assigned the necessary security rights, so that the engines can assert the necessary permissions, if permitted by a given browser's security manager, when read or write operations are required. In one implementation, an installation program is run prior to the first use of the build tools. After installing all of the files, the installation program can install the necessary class libraries required by the run generation process in which the customized and optimized run time engine is created (See FIG. 25). The installation program can also set the necessary environmental variables and installation options.

At 40 the web surfer points a browser at (i.e. calls) an initial build tool HTML/JavaScript file (IBTF). At 41 the IBTF identifies the current browser type and version number. Presently, each browser has different security manager implementations. In one implementation, the invention supports the following three categories:

- 1: With appropriate signing and time stamping, and with appropriate assertions of permissions, the browser will permit local read/write operations.
- 2: With appropriate signing and time stamping, and with appropriate assertions of permissions, the browser will permit local read operations, but write is only legal if sent to a server.
- 3: Local read/write operations are illegal, but are permitted on the server. The IBTF can include a flag that can be set to indicate which security implementation is supported, so that all subsequent read/write operations will comply with the current browser's security manager.

At 42, the IBTF causes the browser to execute the IE so as to sense the screen resolution and for adapting the interface to the user's screen resolution. In one

10

implementation, after entering a delay loop and waiting for the IE to report it is fully loaded and initialized, the IBTF calls two IE methods, which return the width and height of the current screen and browser window. The IBTF then checks for the presence and value of a "mode cookie", to determine whether this is an initialization process, a web site open command process, or a dynamic web page resizing process. If the mode cookie is set to initialize, or it doesn't exist, the IBTF calls the IE to generate the build engine's HTML definition file. At 43 the IE then asserts the required security permission and at 44 creates a build engine HTML definition file and writes this file to the local disk (as appropriate). At 45 the IBTF then turns control over to the PFF for activating the "panel" and build engine and displaying the build engine user interface screen.

The build engine user interface screen includes a "panel" portion and a build engine portion, each of which are loaded into their respective frames, after which the web site page(s) build process can begin. Screen shot FIG. 37 shows a representation of the user interface presented by the build tool. The user interface includes a panel 400 and build frame 500. Panel 400 includes a menu bar 410, menus 420 and sub-menus 430, tool bars 440, status fields 450, interactive fields 460, interactive pull down lists 470 and operational pop-up windows 480. The menu bar 410 can be used for selecting a menu command that will cause a menu to be drawn. The menu (one or menus 420) can be used to select a feature command that could cause an operational pop-up window to be drawn, a direct user input technique or object manipulation technique to be activated, or a sub-menu 430 to be drawn. A sub-menu (one of sub-menu 430) can cause the same type of events as that of a menu. The tool bars 440 include various icons that are shortcuts to feature commands that are also available through the menu bar and its menus. In addition, the tool bar 440 can be used to show the current state of a feature. Status fields 450 show the current value of a certain setting. Interactive fields 460 also show the current value of a setting, but can also be directly changed by the user by typing into the field, with the result immediately processed by the build engine 352 and displayed in the build frame 500. Interactive pull-down lists 470 also show the current value of a setting, but, if selected with a mouse click, will drop down a selection list, which may have an elevator attached. The user can click on an item in the selection list, which will become the current setting with the result immediately processed by the build engine 352 and displayed in the build frame. Operational pop-up windows 480 can have tabs assigned if the number of choices within the pop-up window is large. One or more settings can be changed through a pop-up window, with the results immediately processed by the build engine 352 and displayed in the build frame 500. These interface techniques are described in greater detail below in the build process.

The build frame 500 is used to present the actual web page as constructed by a user. The user can directly enter text, import images, video and audio for display/playback and create animations and transformations that can be viewed in the build frame. FIG. 6, with its attendant screen shots FIG. 38 through 48, shows the user input techniques supported in one implementation of the invention. In one implementation, the user inputs supported include: selection from a JAVA window object (48); selection from a JavaScript window (49) including selection with dual spin control (50*a*) or selection from a JavaScript child window object (50*b*); direct text entry (51); page resizing (52); direct object manipulation (53); and, selection from a JavaScript panel (54).



US 6,546,397 B1

11

In the implementation shown, of the six user input techniques sensed at 13, the code for supporting selections from a JavaScript pop-up window at 49 and selections from the "panel" at 54 were implemented entirely in HTML/JavaScript Code, while support for direct text entry at 51 and direct web page object manipulation at 53 were implemented entirely in JAVA (or any other browser-based full featured programming language). In one implementation, code for supporting selections from a JAVA Window object at 48 and dynamic web page resizing at 52 are implemented using both HTML/JavaScript and JAVA. Those of ordinary skill will recognize that, JAVA could have been used more extensively to implement the methods described at 48, 49 and 54. However, in order to achieve the most intuitive and MS Windows like interface, and because effective two-way communication between JavaScript and JAVA had been achieved (See FIG. 9), the languages proposed appear to best support the particular user input technique.

For example, FIG. 23 shows an actual file selection window 2300, implemented by the invention. This type of file selection window is available in JavaScript/HTML, but not supported by JAVA for applets. File selection window 2300 greatly enhances the interface for the user, as the image, sound clip, or video clip names need not be memorized. File selection window 2300 further eliminates possible operator error when typing in a pathname or filename. The present invention utilized the strengths of JavaScript/HTML with the power of JAVA to create a unique browser based interface solution. In one implementation, the HTML form element "INPUT type=file" was embedded in a JavaScript pop-up window to create the file selection window. The file selection window returns a string value of the image (or other file type) pathname to the pop-up window. The pop-up window's JavaScript then could be used to call a JavaScript function in the panel (panel 400) which:

- 1: Reads the pathname value in the pop-up window.
- 2: Creates a string version of a valid URL by adding the correct URL protocol to the string.
- 3: Updates the panel's database (interface's database 360).
- 4: Calls a JAVA method in the build engine, which casts the string value of the URL into a URL object, creates an image object which is then drawn on the screen, and updates its internal database.

User inputs that are a selection from a JAVA window object (48) permit the implementation of a vast array of intelligent user input interface objects, from sliders to dials, which are extremely intuitive and significantly enhance the user's ergonomic experience. In one implementation, user input interface objects are supported as follows. When a selection from a JAVA window is detected, a pop-up window (applet) is presented (associated with the feature being manipulated, e.g., color, volume) and an engine method is called to begin two-way communication (for passing as arguments any necessary status information). The engine begins polling a JAVA abstract object waiting for a static variable's value to change. The pop-up applet processes the value as defined by a user interaction event, and updates the static variable in that same JAVA abstract object with the new value. Upon detecting a change in the polled static variable, the engine calls the necessary methods to process that new value. These methods include can include a brightness filter that is applied to the image bitmap utilizing techniques very similar to that of that employed by the "fade in" and "fade out" animations, described in association with FIG. 33.

User inputs for a selection from a JavaScript pop-up window (49) can be made in a manner identical to that of

12

making a selection from a dialog box under MS Windows, including the use of tabbed JavaScript pop-up windows. In one implementation when a selection from a JavaScript pop-up window is detected, the panel's (panel 400) JavaScript opens a pop-up window. The pop-up window's initial values are set from a JavaScript database defined in the panel or by the panel calling the engine for the current values and then setting the initial values. In a tabbed JavaScript window, clicking on a tab will call the pop-up window's JavaScript in order to change the state and appearance of the tabbed JavaScript window in the expected way. The pop-up window's JavaScript calls the panel's JavaScript when a completion event occurs. The panel's JavaScript reads or the pop-up window's JavaScript writes the pop-up window's field values, causing the panel's database to be updated, and the panel then calls the appropriate build engine 352 method, passing as arguments the necessary data and status conditions. Initializing the pop-up window's values and updating the panel's database upon completion can alternatively be implemented by JavaScript functions executed within the pop-up window's HTML file.

In addition, there are interface extensions that can extend beyond the usual MS Windows implementations. One is support for a selection from a dual spin control at 50A. Screen shots FIGS. 42-45 show a visualization of an implementation of this interface technique. Screen shot FIG. 42 shows the mouse placed over an upper spin control. Screen shot FIG. 43 shows the result after the user clicked once on the upper spin control. Notice that the value has been incremented by 1, and the text button object is now at a larger point size. Screen shot FIG. 44 shows a combo box list selected by the mouse with the user about to select a significantly larger point size. Screen shot FIG. 45 shows the result of that selection, including the effect on the text button object.

In one implementation, dual spin controls are supported as follows. Each spin control has three visual states, so that when the user places the mouse over the control it appears to light up, and when the mouse button is depressed (pressed down), the spin control is modified to give the appearance of being pressed. JavaScript methods are called in the panel (panel 400) to:

- 1: process each mouse click event over either spin control,
- 2: range check as necessary,
- 3: update the value in the HTML frame object residing in the pop-up window,
- 4: update the JavaScript (panel 400) database,
- 5: call the build engine 352, if necessary, passing the necessary value and status.

If the mouse is clicked on a combo box, the selection window opens in the usual way. If a mouse click in that window is detected, another JavaScript method in the panel 400 is called to update the JavaScript database, and call the build engine 352, if necessary, passing the necessary value and status as function call arguments.

Another interface extension is selection from a JavaScript child window at 50B. This technique helps simplify the number of choices given to the user in a complex pop-up window operation. A selection from a JavaScript child window can be supported as follows. The panel's (panel 400) JavaScript opens the pop-up window. The pop-up window and its child pop-up windows' initial values are set from the JavaScript database defined in the panel 400. The pop-up window's JavaScript opens the child pop-up window and sets its initial values. The child pop-up window's JavaScript calls the pop-up window's JavaScript when a



US 6,546,397 B1

13

completion event occurs. The pop-up window's JavaScript reads the child pop-up window's values, sets those values to its own internally defined variables, and calls the panel's JavaScript. The panel's JavaScript reads the pop-up window's values (which include the settings for its own fields as well as those of its child windows), updates its database, and calls the appropriate build engine 352 method, passing as arguments the necessary data and status conditions. Screen shots FIGS. 46-47 show a visualization of an implementation of a JavaScript child window. Screen shot FIG. 46 shows a change text button style pop-up window. Screen shot FIG. 47 shows the result after the user selected the "Define the Mouse Down Text Button Style" child pop-up window.

Direct text entry is supported at any arbitrary cursor location. In one implementation, "text areas" are utilized in an unconventional way, in order to support full text entry, text editing, text button and paragraph styles, and reformat. Direct text entry can be defined at any arbitrary cursor location, and then text can be dragged to any other arbitrary location.

Text areas are objects that are utilized by JAVA primarily as an interface object for the implementation of a form and are generally "added" to the screen at the initialization time of a JAVA applet. Text areas are decidedly not WYSIWYG. The present invention creates text areas dynamically. Screen shots FIG. 38 through FIG. 41 show a visualization of an implementation of this technique. Screen shot FIG. 38 shows the user selecting a text object from the create text icon object from a tool bar of the panel (panel 400). When the text icon object is selected, the cursor shape is changed to indicate the selection while the text icon object is in the select state. Screen Shot FIG. 39 shows that the cursor has changed shape and that the user has placed the cursor at an arbitrary location on the web page. Screen shot FIG. 40 shows the result after the user has clicked the mouse. A text insertion point and a selection rectangle are drawn at the arbitrary web page location. Screen shot FIG. 41 shows the result after the user has pressed the letter "W" on the keyboard. As can be seen in screen shot FIG. 41, a draw method associated with the build process immediately hides the text area. However, text editor methods associated with the build process continue to utilize the text area as a hidden, dynamically resizing frame, whose size is subject to text button or paragraph style settings, by the amount of text, by the text's orientation (vertical or horizontal) and by the text's font style(s) and font size(s). As the build engine 352 detects a relevant mouse event or keyboard event, the build engine 352 updates the necessary variables that are defined as return values in specified build engine methods. Polling technology (see FIG. 9) retrieves the relevant values and calls the necessary JavaScript method for processing. In one implementation, these same techniques (text area techniques) are used in the scaling technology (See FIG. 30). Since the direct text entry and editing processes bypass completely the interface and the JavaScript code, the polling technology (See FIG. 9) is used to pass the text string values back to the JavaScript database, in order for the interface's pop-up windows to be correctly initialized for subsequent text operations.

Direct text processing at 51 begins with the build engine 352 detecting a "Mouse Drag" or a "Mouse Double Click" event. In one implementation of the present invention, if a mouse drag event is detected, the entire initial anchor word (assuming the "mouse down" event placed the text insertion point within a word) is selected as well as the entire closing anchor word. If a double click event occurs over a word, the entire word is selected. If a double click event is detected

14

over a special hot zone (for example, just to the left of a paragraph line), then an integral number of words are selected. Appropriate four-dimensional variables are set, and a draw system is called. The draw system paints the selected line segment in the marked text background and text color.

The build engine 352 then sets a return flag to be read by the polling technology (FIG. 9). A panel JavaScript poller (FIG. 9) detects this flag and redraws the panel's "Text" menu object showing the choices available when text is selected. In one implementation, the "Text" menu includes choices of "Text Style", "Hot Link", "Preferences", and "Format". The states for the tool bar icon objects of "Bold", "Italic" and "Underline" are set appropriately as is the setting for the point size interactive drop-down list. The panel's JavaScript then calls an appropriate build engine method that resets the flag. If the panel's JavaScript detects the user selecting the "Text Style", "Hot Link", "Preferences" or "Format" choices, it creates the appropriate pop-up window. Upon detecting a user completion event, the panel's JavaScript reads the data settings in the pop-up window, closes that pop-up window, and sends this data to an appropriate build engine method for processing (See FIG. 11). Dynamic web page resizing at 52 is invoked when the build engine 352 detects a user initiated web page resize event. This could be caused by the "Open" or "Web Site" commands from the "File" menu, or from a "Zoom" command from the "View" menu. This technology is explained in detail below in association with FIG. 22.

Direct object manipulation at 53 includes dragging of any object, resizing of non-text objects, rotation and other image manipulation functions, as required. The processing for direct object manipulation begins by analyzing the type of object selected and the state of the object, as set by the interface based on a user's panel selection. The build engine 352 then changes the mouse cursor's appearance, and the type of selection rectangle, including which attachment points, if any, should be drawn and activated. (See FIG. 10 for the mouse event processing technology and FIG. 12 for image processing technology). In one implementation, the same direct object manipulation polling technology is used as described above with regard to direct text entry.

If a selection of an interactive field, interactive drop-down list object, or a toll bar icon object from the JavaScript panel is detected at 54, then the following steps can be invoked, depending on the selection. The point size of a paragraph, a marked text range inside a paragraph or text button object can be changed. The state of an object's 3D frame can be changed. In one implementation, three states for an object frame are supported. The 3D frame can be drawn as a "raised" 3D object, as a "depressed" 3D object, or as a "raised" 3D object that turns into a "depressed" 3D object if a mouse down event is detected over the object to which the 3D frame is assigned. An object's style can be changed. The current web page can be changed. Finally, any other operation that has been defined by a tool bar icon object in the panel can be invoked. This includes the "file" menu choices of new, open and save, the "edit" menu choices of cut, copy and paste, inserting a text, button or image object onto the web page, applying or removing the bold, italic, and underline text attributes for a text or button object, centering or uncentering any web page object, setting the animation for a button or image object, changing the zoom level of the web page, or previewing the web site.

As each new user input is received and processed in accordance with the steps shown in FIG. 6, at all times the internal databases of the JavaScript panel and the build engine 352 are maintained completely in synchronization.

US 6,546,397 B1

15

Synchronization is maintained so that: all status information displayed by the panel is current and correct; all data and status information passed to the build engine 352 from the interface are consistent with the build engine's state at any given time; the values in all pop-up windows are correctly initialized. In order to meet these requirements, all of the variables in the JavaScript panel database are explicitly "typed", to be compliant with the strict variable typing methodology generally imposed in all full featured programming languages such as Java. As JavaScript does not explicitly type anything, where using JavaScript herein, all string, Boolean, and integer variables are typed. Full two-way real time communication support between the JavaScript/HTML interface and the JAVA build engine 352 is provided as described below in association with FIG. 9.

FIG. 7a shows four tools utilized for an implementation of the pop-up window and panel interface technology (15 of FIG. 3). The panel and pop-up windows make extensive use of JavaScript mouse events, including onMouseDown, onMouseUp, onMouseOver, onMouseOut, onClick and onChange methods (56). The pop-up windows make extensive use of the JavaScript onLoad and onUnload methods. In one implementation, when a pop-up window is loaded by the panel, the panel goes into a wait loop, set for 5 times a second using the JavaScript setTimeout method, interrogating in each loop whether the pop-up window's status flag has been set. Meanwhile the pop-up window, when loaded by the browser, executes the onLoad method in order to set a flag in the panel informing the panel that the pop-up window is now loaded. Upon detecting the load event completion, the panel then proceeds to initialize the fields in the pop-up window. The panel will always close a pop-up window after detecting its completion event. However, if the user has closed the pop-up window in a non-standard way, the pop-up window executes the onUnload JavaScript method, which sets a flag in the panel notifying it that the pop-up window has been closed.

The JavaScript code in the panel and in all pop-up windows make extensive use of JavaScript method onKey-Down for the following operations:

- 1: When the focus is on the icon representing completion ("OK" is used in many MS Windows applications) causing the enter key to initiate a pop-up window/panel completion event.
- 2: When the focus is on the icon representing cancellation ("cancel" is used in many MS Windows applications) causing the Esc key to initiate a pop-up window/panel cancellation event.
- 3: When the focus is on any pop-up window or panel object, such as a data entry field, a check box, a radio button, a drop-down and scrollable list, a scrollable list, an icon, or a DHTML tab object (discussed below), the navigation keys are captured by the onKeyDown method, a JavaScript function is called, and the appropriate change is made. For all pop-up window and panel objects, when the Tab key or the combination of the Tab key with the Shift key are detected, the onFocus JavaScript method is employed and the focus moves to the appropriate pop-up window object. If the pop-up window or panel object is a data entry field, drop-down list or a scrollable list, all cursor key operations are detected and the insertion point is adjusted accordingly. If the pop-up window or panel object is a check box, radio button a icon, or a DHTML tab object, and a cursor key (up, down, left, right, home and end keys, with or without the Ctrl or Shift keys) is detected, the onFocus JavaScript method is employed and the focus moves to the appropriate pop-up window object.

16

One methodology for this feature requires that all keyboard events be monitored, at all times. When the scan code for the enter key is detected, the appropriate JavaScript function is called to close a pop-up window and to call the appropriate JavaScript function for processing of the relevant data (updated in the window) and communicating, as necessary, with the build engine 352. In another implementation, rather than the panel going into a wait loop awaiting notification from the pop-up window for data initialization purposes the pop-up window, when loaded, executes the onLoad JavaScript method, and reads the required data values directly from the panel's database, utilizing the JavaScript "opener.fieldname.value" technique. Similarly, the pop-up window, when detecting its completion event, updates the panel's database with the revised values from its own fields and then calls the appropriate JavaScript function in the panel for further processing. Both implementations, and any combinations, assure that the pop-up windows are correctly initialized, the panel's database is correctly updated, and the data is successfully sent to the build engine 352 for processing.

Extensive use of JavaScript technology is employed to enhance the user interface and for communication between the various HTML frames and/or files, within a given HTML frame or file, between an HTML frame and the JAVA engine, and as a bridge between two different JAVA applets (57). Extensive use is made of JavaScript arrays to store the values of all page and object attributes, to initialize the correct values in all pop-up windows, and to pass data and status to the engine. Various JavaScript techniques are employed to "type" all variables (JavaScript does not explicitly type anything as described above) as a prerequisite for passing values to the build engine 352. Variables that should be typed as strings, integers and Booleans are typed through the use of "Eval" and "New" JavaScript functions. The choice of color, found in most pop-up windows to define one or more color elements, can be implemented utilizing several innovative JavaScript techniques. They include:

- 1: Defining a complex image map through a JavaScript function utilizing arrays. Screen shot FIG. 48 shows a visualization of an image map. A JavaScript computational loop utilizing arrays can be used to define each individual rectangle in this color palette with its appropriate RGB value and a function call to the appropriate JavaScript method.
- 2: Limiting the color choices from the image map to only those colors that are designated as safe colors. Safe Colors are the subset of all colors that are browser independent, assuring a consistent color look across all browsers.
- 3: Supporting a dual color selection technology. The user can be presented with a color palette and can click on a particular color in the color palette. Image map technology can call a JavaScript function, which converts that choice into a RGB numeric definition. This definition updates the RGB values shown in screen shot FIG. 48, as well as passing those values, through an appropriate function call, to a build engine JAVA method. The build engine 352 will then draw the actual color immediately on the web page. Alternatively, the user can select a value from Red, Green or Blue selection lists, which can be implemented using an HTML drop-down list form object. The value selected is then processed by an appropriate JavaScript function call to a build engine method, which converts the RGB to a JAVA compliant value, and then draws the actual color on the web page.

US 6,546,397 B1

17

4: Supporting True Transparency. For appropriate color elements, such as the background for a text button object, the user can choose, either from the color palette by clicking on a "transparency" rectangle, as described above, or by selecting "TTR" from a Red, Green or Blue selection list. This choice is then processed by an appropriate JavaScript function call to a build engine method, that in turn sets a particular flag for the draw system (of the Build Tool) to not draw a background color for that object.

Innovative techniques are used to enable JavaScript to dynamically create HTML code based on real time conditions. Cookies can be used for data communication between HTML frames and HTML files, some of which were created in real time. Many unique combinations of HTML elements, including frames, forms, and tables, enhanced by JavaScript code, are utilized to create extensions beyond that of the MS Windows interface (58). For example, a dual combo box/spin control for both small and large numeric incremental jumps can be implemented by a combination of form and table elements, mouse events, and JavaScript methods.

Extensive use of Cascading Style Sheets (CSS) was employed to create a consistent look for all pop-up windows, and for precision placement of various HTML elements (59).

FIG. 7b shows a detailed view of the build-time techniques for implementation of panel interface objects, including the menu bar, menus and sub-menus, the tool bars, status fields, interactive fields, and interactive pull down lists, in accordance with one implementation of the present invention (15 of FIG. 3). These techniques create panel interface objects that have the same look and feel of those which are implemented under the various MicroSoft Windows Operating Systems. In one implementation of the present invention, the status fields, interactive fields, and interactive drop-down lists are defined as HTML form objects (text boxes and lists) embedded within DHTML objects. The menu bar, menus and sub-menus, and the tool bars can be defined as pure DHTML objects. However, Cascading Style Sheets can be used for all panel interface objects; although more extensively with DHTML objects as will be described below. In an alternative implementation of the present invention, the status fields and interactive drop-down lists are defined as pure DHTML objects.

In one implementation of the present invention the menu bar at 270 is defined as sets of DHTML objects, each set corresponding to a menu command. Each set consists of four DHTML objects with absolute screen positioning, one defining the DHTML object in the Mouse Over state at 278, the second for the Mouse Down state at 279, the third for the Active state, and the fourth for the Inactive state. Each state has a different CSS style assigned, which defines the visual appearance of that state. When the build tool is initialized at FIG. 5, the appropriate menu commands are initialized as active or inactive at 277. If the menu command is defined to be inactive, that DHTML inactive object is assigned by a JavaScript function to the "visible" style attribute, while the other three DHTML objects are assigned the "hidden" style attribute. Screen shot FIG. 38 shows a visualization of the "Interactions" menu command in the inactive state. In the inactive state all user interactions are ignored. If the menu command is defined to be active, that DHTML active object is assigned by a JavaScript function to the "visible" style attribute, while the other three DHTML objects are assigned the "hidden" style attribute. While in the active state, the JavaScript functions for "onMouseDown", "onMouseUp", "onMouseOver" and "onMouseOut" are implemented. If a

18

Mouse Down user interaction event is detected over an active menu DHTML object at 279, a menu command specific JavaScript function is called. This function sets the DHTML object for the Mouse Down state to the "visible" style attribute, calls a generalized JavaScript function to reset the visibility states all the other appropriate DHTML objects, set certain status variables, and set the DHTML object which defines the menu associated with that menu command to the "visible" style attribute. Screen shot FIG. 37 shows a visualization of the "Image" menu command after having received a mouse down event, with its associated menu 420 having been set to the "visible" style attribute. If a mouse up user interaction event is detected over an active menu DHTML object at 281, a generalized JavaScript function is called in which the DHTML object defining the mouse over state is passed as a function call argument. This function sets the DHTML object defining the mouse over state to the "hidden" style attribute thus resulting in the appearance as shown for the image menu command in screen shot FIG. 37, even when the mouse has been moved off the menu object. If a mouse over user interaction event is detected over an active menu DHTML object at 278, a generalized JavaScript function is called in which three DHTML objects are passed as function call arguments as well as a menu command name. These DHTML objects are the ones defining the mouse over state, the mouse down state, and the associated menu. This JavaScript function first tests to see if a menu has been activated by a previous mouse down event and is still visible. If so, a generalized "reset visibility states" function is called, then both the mouse down and associated menu objects are set to visible. Finally the same menu specific function is called as with the mouse down event. If no menu is visible, then the object associated with mouse over state is set to visible. If a mouse off user interaction event is detected over an active menu DHTML object at 281, a generalized JavaScript function is called in which the mouse over DHTML object and the menu command name are sent as arguments. Logic tests are made to determine which menu command object has been left, as well as whether any menus are currently visible. Depending upon the results, the mouse over DHTML object may be set to hidden.

In one implementation of the present invention the menus and sub-menus at 271 are defined as a set of DHTML objects, one for each menu choice, nested inside an DHTML object that defines the entire menu. Each menu object is given absolute positioning, while the menu items are given absolute positioning relative the menu objects origin. Both the entire menu and each choice are assigned CSS styles to define their visual appearances. For each menu choice the JavaScript functions for "onClick", "onMouseOver" and "onMouseOut" are implemented. If a mouse click event is detected at 280 and no sub-menu is defined, a feature specific JavaScript function is called. First the menu bar and the menus are set to their appropriate visibility states. Then setting their visibility attribute style to "visible" activates the appropriate tool bar icon DHTML objects. Finally the feature specific JavaScript code is executed as discussed herewithin, which may cause a pop-up window to be displayed, the Panel's database to be updated, and/or the build engine 352 to be called. If a mouse over event is detected at 278 and no sub-menu is defined, a generalized JavaScript function is called in which the menu choice object is passed as an argument. This function first calls a generalized JavaScript function to close any pop-up windows, then set a status variable and finally executes DHTML commands to set the correct text and background



US 6,546,397 B1

19

colors for the object. If a mouse off event is detected at 282 and no sub-menu is defined for a menu choice either immediately above or below, a generalized JavaScript function is called in which the menu choice object is passed as an argument. A status variable is set and DHTML commands are executed to set the correct text and background colors for the object. If a sub-menu is defined for a menu choice object, then the same sub-menu specific JavaScript function are called for both mouse click or mouse over events. This function performs the same steps as that of the generalized function that was called for a mouse over event, as well as setting the sub-menu object and its menu choice objects to the visible state. Screen shot FIG. 37 shows a visualization of the menu bar's "Image" command having been activated, the drawing of its associated menu 420, the selection of the "Enhance" menu choice, and the drawing of the "Enhance" sub-menu 430. In the event that the cursor is moved to an adjacent menu choice under the "Image" menu, such as "Animation . . ." or "Rotate", then a specific JavaScript function is called which, in addition to the functions executed by the generalized JavaScript mouse over function, also hides the "Enhance" sub-menu.

In one implementation of the present invention, the tool bars at 272 are defined as a DHTML objects, and a set of DHTML objects are defined for a tool icon. The tool is given absolute positioning and is assigned a CSS style in order to define its visual appearance. Each tool icon is assigned a set of three DHTML objects all with absolute screen positioning. The first DHTML object defines the mouse over state at 278, the second for the mouse down state at 279, and the third for the active state. Each state has a different CSS style assigned, which defines the visual appearance of that state. For each tool icon active state object the JavaScript functions for "onClick", "onMouseDown", "onMouseUp", "onMouseOver" and "onMouseOut" are implemented. GIF images are defined for the tool bar DHTML objects, and may be always visible. The inactive "grayed out" representations for each tool icon can be drawn on this image. When the build tool is initialized at FIG. 5, the appropriate tool icon objects are defined as active or inactive at 277. The inactive state for an tool icon is represented when all three of its associated objects are assigned the visibility style of "hidden". Screen shot FIG. 38 shows a visualization for several inactive tool icons including the icon commands for bold, italic, underline, left and centered. All user interaction events are ignored in the inactive state. If the tool icon, based on the state of the build engine and based on the polling technology described below, is set to an active state, then the tool icon's active state object is set to the visibility style of "visible". If a mouse click event is then detected at 280, a feature specific JavaScript function is called in a manner identical to that for a mouse click event over a menu choice object as described above. If mouse down or mouse up events are detected at 279 or 281, then respective generalized JavaScript functions are called, in which the DHTML object defining the mouse down state is passed as a function call argument. If a mouse down event was detected, then the generalized function sets the tool icon's mouse down object to the "visible" state. If a mouse up event was detected, then the generalized function sets the tool icon's mouse down object to the "hidden" state. If mouse over or mouse out events are detected at 278 or 282, then respective generalized JavaScript functions are called, in which the DHTML object defining the mouse over state is passed as a function call argument. If a mouse over event was detected, then the generalized function sets the tool icon's mouse over object to the "visible" state. If a mouse off event was detected, then the generalized function

20

sets the tool icon's mouse over object to the "hidden" state. Screen shot FIG. 37 shows a visualization of the button tool icon with both its associated mouse down and active objects set to "visible". Screen shot FIG. 38 shows a visualization of the text tool icon with both its associated mouse over and active objects set to "visible".

In one implementation of the present invention, the status fields at 273 and the interactive fields at 274 are defined as HTML text boxes. In an alternative implementation status fields are defined as DHTML objects. For both of these panel interface object types, under certain conditions, the panel draws status information into these panel interface objects. This information can result from user input as discussed in FIG. 6, or through the polling and two-way communication technology between the interface and the build engine 352 as discussed below. In one implementation of the present invention the status fields are:

- 1: The color of the selected web page object, in which the red, green and blue settings are shown.
- 2: The animation state of the selected button or image object.
- 3: The zoom level for the current web page.
- 4: The point size for the selected text or button object.
- 5: The horizontal position, in pixels, of the mouse cursor.
- 6: The vertical position, in pixels, of the mouse cursor.
- 7: The type of web page object (text, button, image, table, form object, draw object, etc.) if selected. The type of object that the mouse is over, if no object is selected.
- 8: The width, in pixels, of web page object (text, button, image, table, form object, draw object, etc.) if selected. The width of the object that the mouse is over, if no object is selected.
- 9: The height, in pixels, of web page object (text, button, image, table, form object, draw object, etc.) if selected. The height of the object that the mouse is over, if no object is selected.

Screen shot FIG. 38 shows a visualization of the status fields in one implementation of the invention 450. In an alternate implementation using DHTML objects, the status fields will appear two-dimensional rather than the three-dimensional look currently shown.

There is one interactive field defined in one implementation of the present invention. Screen shot FIG. 37 at 460 shows a visualization of the page number interactive field. In addition to the current web page being displayed, either as a number in one implementation or as a user defined name in an alternative implementation, the user can place the cursor into this field and enter the desired page to go to. A click at 280 or Enter Key event will execute this function.

In one implementation of the present invention, the interactive drop-down lists at 275 are defined as HTML form lists. In an alternative implementation, status fields are defined as DHTML objects. For both of these panel interface object types, under certain conditions, the panel draws status information into these panel interface objects. The interactive drop-down lists behave in a manner very similar to that of interactive fields, except that when selected, a selection list drops down for selection. Depending upon the number of entries in the list, an elevator object may be drawn. The operations of selecting the interactive pull down list, the selecting of a list item, or the operation of the elevator is identical to that of comparable MS Windows objects. In one implementation of the present invention the interactive pull down list are:

- 1: Zoom. This interface object has dual spin controls as described above and is always selectable except when in a preview mode. It shows the current zoom level.

US 6,546,397 B1

21

- 2: Button Style. This interface object is always selectable except when in preview. It shows the button style of the currently selected button, if any. Changing the button style will change the style of the currently selected button, and/or define the style of the next button to be created.
- 3: Point Size. This interface object has dual spin controls as described above and is selectable when a text or button object is selected. It shows the point size of the currently selected text or button object, if any. Changing the point size will change the point size of the currently selected text or button object.
- 4: Paragraph Style. This interface object is always selectable except when in preview. It shows the paragraph style of the currently selected paragraph, if any. Changing the paragraph style will change the style of the currently selected paragraph, and/or define the style of the next paragraph to be created.
- 5: Frame State: The state of the 3D frame (none, raised, pressed or live) of the currently selected text, button, or image object.
- 6: Image Style. This interface object is always selectable except when in preview. It shows the image style of the currently selected image, if any. Changing the image style will change the style of the currently selected image, and/or define the style of the next image to be created.

Screen shot FIG. 37 shows a visualization of interactive drop-down lists 470. In an alternate implementation using DHTML objects, the interactive drop-down lists will appear two-dimensional rather than the three dimensional look currently shown.

Tool bar icon objects, status fields, interactive fields, and interactive pull down lists all show feedback of the current build engine state. The technology utilized by one implementation of the invention is described below.

FIG. 7c shows a detailed view of the of the build time techniques for implementation of tabbed pop-up windows (15 of FIG. 3). These techniques create a pop-up window interface that visually and behaviorally is identical to that which is implemented as dialog boxes under the various MicroSoft Windows Operating Systems. Pop-up windows can be non-tabbed as described in FIG. 7a, or can have from two to as many as 10 or more tabs, depending upon the complexity of the choices available to the user for a given feature. In one implementation of the present invention each tab at 283 is defined as a DHTML object at 284. The tab is given absolute positioning and is assigned a CSS style at 286 in order to define its visual appearance. When a click is detected through the JavaScript "onClick" function, a tab specific JavaScript function at 285 is called within the pop-up window's HTML file. This function sets the display style attribute for the DHTML objects that define the settings for all the non-selected tabs to the display style attribute of "none". The DHTML objects that define the GIF image of the non-selected tab file representations are also set to the display style attribute of "none". The display style attribute for the DHTML objects that define the settings of the currently selected tab and the GIF image that depicts the selected tab file representation is set to the display style attribute of "". If there is to a change of the focus of the selected field within the now to be visible tab specific choices, the focus attribute for that object is executed. Screen shot FIG. 37 shows a visualization of a tabbed pop-up window, and screen shot FIG. 63 shows a collage of four views of a tabbed pop-up window with four tabs. Notice that each state of the tabbed pop-up window has a different tab file representation, showing the selected tab as being in the forefront.

22

The interface technology of the invention, in addition to its utilization as part of a web-based web site generation tool, can be used to provide a general purpose interface for all web-based applications that want a MS Windows compliant interface.

A process for updating the internal database of the build engine 352 is shown schematically in FIG. 8. The database is compact and efficient in order to meet the performance requirements for the run time process. The database handles a wide selection of data objects, including multi media objects such as colors, fonts, images, sound clips, URLs, threads, and video. The database supports a multi level animation, transformation, and time line model (discussed in greater detail below). The database complies with the differing rules imposed by the various popular browser security managers.

The process begins by determining the type of data to be updated at 60. Data that defines generic web site settings (See FIG. 21a), screen resolution values (See FIG. 21a and FIG. 24), and the web page high watermark setting (See FIG. 24) can be stored in a header record as boolean and integer variables and URL and color objects at 62 and 63. Data that defines web page, paragraph, text button, and image style and text button, image and paragraph high watermark settings can be stored in one-dimensional arrays as boolean, integer and string variables and URL, font, image or thread objects at 61 and 64. The URL, color, font, image and thread objects can also be created as required at 64.

Data that defines text button, image, paragraph, or other parent objects and paragraph line high watermark settings can be stored in two-dimensional arrays (by web page and by object number) as boolean, integer, string, floating point variables and URLs at 65 and 66. Again, the URL, color, font, image, audio clip, video clip, text area and thread objects can also be created as required at 66. Data that defines a paragraph line and paragraph line segment high watermarks can be stored in three-dimensional arrays (by web page, by paragraph number, and by line number) as Boolean, integer or string variables at 67 and 68. Again, the URL, color or font objects can be created as required at 68. Data that defines a paragraph line segment can be stored into four-dimensional arrays (by web page, by paragraph number, by line number and by line number segment) as Boolean, integer or string variables or URL, color and font objects at 67 and 68. As a data field is added, changed or deleted, a determination is made at 69 on whether a value for a given high watermark needs to be changed. If so, it is updated. As a specific method in the build engine is called, a determination is made at 70 on whether a feature flag needs to be set. For example, if a particular JAVA method is called, which requires an instance of a certain JAVA Class to be executed by the run time engine, then that JAVA Class is flagged, as well as any supporting methods, variables and/or object definitions. The use of these flags is described in greater detail below in association with FIG. 25 and FIG. 27 to create a compact and efficient customized run time environment.

In one implementation, the header record, the style record, the web page record, and the object records, are carefully defined in a specific order, written in that order, and explicitly cast by object type when read by the run time engine. Exception handling can be implemented to recover from any errors. This helps assure that data integrity is maintained throughout the build and run time processes.

FIG. 9 details the polling process (16 of FIG. 3). The polling technology is essential for creating the necessary

US 6,546,397 B1

23

two-way real time communication between the JavaScript/HTML interface and the JAVA build engine. Since there is no particular difficulty for JavaScript to be able to call and pass values directly to JAVA methods, the technological challenge is to find a reasonable technique to enable JAVA to communicate back to JavaScript. The polling technology is generic, and workable across all the current browsers. The polling technology is flexible, as there are no real constraints as to what data could be communicated from the build engine to the interface, and this communication can occur at any time. The polling technology is reasonably efficient, so that the performance of the build process is not significantly affected.

In one implementation, two different techniques were utilized to implement this capability. The first was to place the build engine inside a JAVA wrapper. The JAVA wrapper accepts direct communication from JavaScript function calls, interrogates a particular JAVA build engine method, and returns that method's return value back to the calling JavaScript function. The second technique was more unconventional. A polling loop is defined in the panel's (panel 400) JavaScript that creates a near continuous, at least from a human perception point of view, dynamic real time link, in order to monitor events occurring inside the build engine. The result is a real time retrieval (from an ergonomic perception point of view) of necessary data and status settings from the build engine back to the interface.

Upon the loading of the panel HTML file, a JavaScript function at 71 (the poller) is immediately called which initiates a polling loop. In one implementation, the polling loop is set at a poll rate of once every 100 milliseconds or less. The polling routine, operating through the JAVA wrapper, calls the build engine in order to read the current horizontal and vertical coordinates of the mouse cursor, and displays them in the panel's status fields (FIG. 37 at 450). The polling routine also polls the build engine in order to detect whether the mouse has moved over a valid object or, by inference, whether a mouse single click, or double click event has occurred. The poller is also constantly requesting the JAVA wrapper to return the status of an error flag in order to inform the user, if necessary, of an unrecoverable error condition, and the reason for it. (See FIG. 10). The poller then calls a panel JavaScript function that, in turn, calls the build engine to reset the flag. The poller constantly requests that the JAVA Wrapper return the status of whether the mouse cursor is over a valid object, and, if so, that object's number, type, height and width. The poller also constantly requests the JAVA wrapper to return the status of whether an object is selected, and, if so, the type and number of that selected object, as well as the objects height, width, and 3D frame state (and the point size of the object's current font if the object is a text button or paragraph object). In addition, if the object is a paragraph, the poller constantly requests the JAVA wrapper to return a flag if a double click or drag mouse event has occurred.

At 72 the polling routine detects a mouse event based on analyzing the return values received. The poller can infer that the mouse has either moved off or moved on to a valid object at 73 if the mouse over object state has changed or the mouse over object number has changed. If so, the poller updates the relevant interface objects of the panel as appropriate and displays them as necessary, and, depending upon whether the object is a text button object, a paragraph, image object, etc., at 75, begins polling their unique values.

The poller can infer that a single click mouse event has occurred at 74 if the selection state has changed, or the selected object changed. The poller updates the menu bar

24

(FIG. 37 at 410) as appropriate, making the appropriate menu commands either active or inactive. The poller also sets the necessary status variables, and, depending upon whether the newly selected object is a text button object, a text object, image object, etc., at 74, begins polling their unique values. The poller also activates the appropriate menu choice objects inside the "Edit" menu, the "Text" menu, the "Button" menu, the "Image" menu, and the "Interactions" menu objects (FIG. 37 at 420 and 430), depending upon whether an web page object is selected or not, which type of web page object is selected, or, if the selected web page object is a text object, whether text is marked through a drag or double click event. In a similar manner, the poller also sets the values for the interactive field objects (FIG. 37 at 460) and the interactive drop-down list objects (FIG. 37 at 470). More specifically, JavaScript can poll the web page object number. The value of the web page object number can be used to initialize pop-up windows with that object's web page current values, either from the panel's database or, if necessary, by interrogating the build engine's database.

The poller can infer that a double click or mouse drag operation has occurred if the flag indicating a double click or mouse drag operation is detected at 75. The poller calls a panel JavaScript function that, in turn, calls the build engine to reset the flag. The poller then calls a panel JavaScript function to display the appropriate panel menu choices. For example, if the double click or mouse drag event occurs within a text object, then the "Text Style" and "Hot Link" menu choice objects become active under the panel's "Text" menu object.

Depending on the object type (76), the polling technology performs various functions. If the object is a text object at 77, the values for the paragraph style, point size, object height and width, text color, and the 3D frame status are polled and displayed. The panel's menu objects and the menu choice objects within that are active for a text object are set to the active state, and the non-text menu choice objects are set to the inactive state, which visually means they are unavailable and are "grayed out". In addition, polling can be initiated for the creation of a hot link. If the object is a text button object at 78, the values for the text button style, point size, object height, width, text color, animation state, and 3D frame status are polled and displayed. The menu choice objects inside the panel's menu objects that are active for a text button object are set to the active state, and the non-text button menu choice objects are set to the inactive state, which visually means they are unavailable and are "grayed out". The value of the text button object string is also polled and saved in the panel's database for use when initializing relevant pop-up windows. If the object is an image object at 79, the values for the image style, object height, width, frame color, animation state, and 3D frame status are polled and displayed. Again, the menu choice objects inside the panel's menu objects that are active for a image object are set to the active state and the non-text button menu choice objects are set to the inactive state. In addition, the results of any relevant direct object manipulation are polled and displayed.

FIG. 10 describes a two level error correction technology (17 of FIG. 3) employed by the build process. Initial error checking occurs during the interactions between the user and the interface with the JavaScript error checking code at 80. Any file name, selected by the user through the file selection window or typed in a file pathname (See FIG. 6 at 49) is checked by the panel's JavaScript to assure that it has the correct file type suffix (gif, jpg, au, etc.) at 81.



US 6,546,397 B1

25

The panel's JavaScript Code performs range checking at 82 to prevent user error or to prevent the breaking of any internal limits imposed by the build engine. These can include: going to a non-existent web page; exceeding any limit with the dual spin control (i.e. attempting to increment or decrement a point size outside of the legal range, or trying to illegally decrement a value to zero or a minus number; typing in a numeric value that is outside a legal range; and, implicitly creating an object that exceeds a limit imposed by the build engine).

The panel's JavaScript code also checks the file pathname to make sure it contains a valid address, and makes necessary additions or conversions, if necessary, at 83. For example, if the user selected a file from the local disk, the correct URL protocol is appended to the file name in order to make it a valid string representation of a URL address. Any illegal characters for a pathname or a null file pathname entry are also caught at 83. In addition to file pathname validity checking there are other validity checking functions that can be employed by the JavaScript at 83. They include the attempt by the user to enter a non-numeric character into a numeric field, or leaving an essential fill-in field empty.

The panel's JavaScript then passes these values to the build engine through the arguments of a JAVA method function call at 84. The build engine can utilize the extensive exception handling capability of JAVA at 85 (or that of any other full featured programming language used) to attempt to recover from any processing error. If recovery is not possible, the build engine sets an error flag, utilizing the polling technology (See FIG. 9 at 71). The poller, upon detecting this flag, informs the user, for example, through an alert JavaScript pop-up message, what non-recoverable error has occurred, from which operation, and what actions, if any, the user should take. For example, if the user had selected a corrupted image file, the exception handling technology can inform the user of this fact so that user corrective action can resolve this very common problem. In one implementation, error handling and exception recovery support is provided for a malformed URL, an input or output error, a security manager violation, and a null pointer error.

FIG. 11 shows a process for text entry and text processing (18 of FIG. 3). The process begins when the panel's JavaScript detects the user selecting either "Button" or "Text" icon objects from the panel's tool bar or from their equivalent menu choices under the "Button" or "Text" menus, and calls the appropriate JavaScript function at 86. The JavaScript function, after performing a range check to assure that no internal limits of the build engine are being broken, updates its database, and sets the necessary status variables. The panel's JavaScript then calls the appropriate build engine method, passing the necessary arguments, including the current board number, the internal number to be assigned to the object, the object type, and the current text button or paragraph style at 87. The build engine then updates its internal database and sets the necessary status variables. The build engine also changes the mouse cursor shape to that of a text entry symbol. In one implementation, the mouse cursor is shaped like a crosshair, and can be moved onto the web page (the build frame 402) at an arbitrary location.

The build engine detects a mouse click event through its "mouseDown" method at 88. This method reports to the build engine the exact horizontal and vertical coordinates of the crosshair mouse cursor at the moment the mouse button is pressed. The build engine places these values into its internal database. The polling process is also supported, as discussed in FIG. 9, by placing the necessary return values in the appropriate poll enabled methods.

26

The build engine creates a dynamically resizable frame utilizing JAVA's "Text Area" object class, whose coordinates and size coincide with that of the draw system for the object as defined below. Other full-featured programming languages, if used by the invention, also possess similar object types. The text area is immediately overdrawn by the draw system's background paint routine. The build engine, utilizing the font metrics as defined by the selected text button or paragraph style, and utilizing the crosshair cursor's coordinates, calls the draw system. The draw system paints the background and then paints an insertion point and a selection rectangle, in the appropriate colors, and with the appropriate height and width, into the appropriate web page location at 89. If the text button or paragraph style has a 3D frame selected, this intelligent ornamental object would also be drawn, in the appropriate color, dimensions, and thickness. Screen shot FIG. 41 shows a visualization of this process. The text insert point is in black, surrounded by a red selection rectangle, and surrounded by a blue 3D frame, as defined by the selected style. The text editor is then initialized by setting the necessary status variables.

The build engine waits until a keyboard keystroke is detected. The scan code is interpreted, and if it is a text entry key, the text editor's methods are called at 90. The text editor processes the key event at 91. The build engine employs frame (Text Area) processing methods and draw methods to implement the text entry and text processing functions. As a keyboard key for a text character is pressed, the build engine passes this value to the editor's text entry method, which updates both the text area's frame definition, and the draw system's database. The width of the text area is dynamically resized as necessary. If the object was a paragraph, a check is made on whether a reformat event should occur, based on the paragraph style's definition and the width of the current line's text string. If so, the appropriate text editor reformat method is called, which may cause the text area's vertical dimension to also be resized. A high watermark variable may also be set, for optimization purposes. After the final state of the text area is determined for the text entry keyboard event, the internal database for the text area, and for the paragraph or text button object, are updated. The draw system is called, and the results of the text entry event are drawn on the web page at 94.

In one implementation, the build engine also supports the usual text processing functions found in MS Windows or Macintosh based Word Processors or Desktop Publishers at 92 and 93. For example, if the user single clicks the mouse when over an unselected paragraph or text button object, that object is selected, a selection rectangle is drawn, the mouse cursor shape is changed to a crosshair, and the poller reports the necessary information to the panel's JavaScript. If a mouse click occurs over a selected paragraph or text button object, the editor's "Set Text Insertion Point" method is called. Based on the coordinates of the mouse cursor, and based on a calculation by the build engine as to the nearest line, and the nearest character on that line, the text insertion point can be drawn appropriately, and the necessary status variables are updated. Text entry is then processed as discussed at 91.

If a double click or mouse drag mouse event is detected over a paragraph, an appropriate "text string selection" method is called (See FIG. 6). Based on the coordinates of the mouse cursor, and based on a calculation by the build engine as to what text string should be selected, the internal database is updated, appropriate status variables are set, and the draw system is called for marking the text string at 94. The polling technology is activated as discussed in FIG. 9.

US 6,546,397 B1

27

The build engine's reformat methods for paragraphs can utilize a "Clean Text Stream" model for calculating line breaks and for updating four-dimensional variables utilized by the draw system in order to draw each paragraph, each paragraph line, and each paragraph line segment in the correct location, with the correct font type, font style, font size, font effect, and background and text string color. Font style refers to a font format such as Normal, Bold, Italic, or Bold Italic; Font effect refers to style overrides such as Underline, Double Underline, Small Caps, Cross Out, Superscript, Subscript, etc. The "Clean Text Stream Model" implemented by the build engine maintains multi-dimensional array pointers and records for every paragraph line and line segment external to the text string defined within the text area. Three-dimensional and four-dimensional variables are updated after each text entry or text editing and processing event in order to assure that the pointers into the paragraph text stream, defined in the text area, are current. The three-dimensional variables that the build engine has implemented can include soft and hard line end pointers for each paragraph line. Their values can be the absolute character positions within the text area text string for that line end. Hard line breaks can be created by the user pressing the enter key. Soft line breaks can be created by a reformat method based on a calculation defined below.

The four-dimensional variables can be absolute pointers into the text area text string for the beginning and end of every style override, associated with each paragraph line segment. These style overrides can include hot links, font type, font style, font size, numerous font effects, and text and background colors. For each style override there is an associated style override record that maintains all the font and color settings for that paragraph line segment. Also positional and size data such as start and end pointers into the paragraph text stream, a left offset relative to the paragraph's left origin, a top offset relative to the paragraph's top origin, and the line height. The style override record is created when the build engine detects a mouse drag or mouse double click event within a selected paragraph. When the mouse button is initially pressed, the current paragraph line and current word on that line are calculated in a manner identical to that for calculating the location of the text insertion point on a mouse click operation. The entire word becomes one anchor for the paragraph line segment, while the word defined by the mouse coordinates when the mouse button is released becomes the other anchor. Up to two other paragraph line segments can be implicitly created by the word oriented selection method. If there is text to the left of the first anchor word, and that paragraph line had not previously had a style override defined in it, the text string from the beginning of the paragraph line to the first anchor point has a style override record created for it. The values are set to that of the underlying paragraph.

If style overrides had already been created on that paragraph line, and the anchor word is inside one of them, then that style override's end pointer is adjusted to the start of the anchor word. All other style overrides, if any, to the right of the anchor word are deleted, as overlapping style overrides are not permitted. In a similar manner, the text string, if any, to the right of the last anchor point, up to the line or paragraph end, can also be defined as a style override. If a mouse click occurs before a "text style" operation, then these pointers will be reset. If the panel's JavaScript detects a user selection of "text style" from the "Text" menu, the appropriate pop-up window is drawn and its values initialized from the JavaScript database. Upon detecting a user completion event (i.e., the depressing of the enter key), the

28

panel's JavaScript database is updated and a call is made to an appropriate build engine method, with the necessary data and status information passed as function call arguments. The build engine updates its internal database and calls the reformat method if necessary. The draw system utilizes these four-dimensional variables in order to paint the paragraph line segment style override.

The calculation for the creation or updating of a soft line break begins with the maximum paragraph width, which is set at a percentage of the browser screen width. This percentage is converted to an absolute pixel number based on the web designer's screen resolution. When any text entry or text editing and processing event occurs, a build engine method is called which calculates the width, in pixels, for the current paragraph line, based on the character string in the text area that exists between the previous line end pointer and the current line end pointer. The font definition(s) that are related to this character string are applied, and a string width is calculated. If the string width exceeds that of the maximum paragraph width, an "Overflow" reformat method is called. The overflow reformat method calls a method to determine the settings for the last word on that line, and that word overflows to the following paragraph line. All pointers for the current line, and subsequent lines are updated as necessary, as are all pointers and records to paragraph line segments. If the string width is less than that of the maximum paragraph width, and the text processing operation was not text entry, then an "UnderFlow" Reformat method is called. The underflow reformat method calls a method to determine the width, in pixels, for the first word on the next line. If that word will fit on the current line it is placed there. As before, all pointers for the current line, and subsequent lines are updated as necessary, as are all pointers and records to paragraph line segments. The word oriented selection technique, and the reformat, database, and draw technologies that support it, greatly simplify the text editor and produce a run time engine that is smaller, faster and more reliable.

FIG. 12 shows the operation of the image processing technology utilized by the build engine (19 at FIG. 3). The process begins when the panel's JavaScript detects the user selecting the "Image" icon from the panel's tool bar or the comparable menu choice under the "Image" menu. The appropriate JavaScript function is called at 95, which draws the define image pop-up window. The user then selects an image from the file selection window with the browser, types in the image pathname for the image file on the local disk, or types in the URL for the image that exists on a server. The user could also define a 3D frame for the selected image at this time. Screen shot FIG. 49 shows a visualization of a collage for the define image pop-up window and the user's selection choices under each tab setting. The user can complete the selection process by either pressing the Enter Key or clicking on the "Create Image" icon in the pop-up window. If the Enter Key is pressed, the pop-up window's JavaScript Code utilizes the onKeyDown function, or if a mouse click, the onClick function, as described in FIG. 7, to recognize the completion event. An appropriate error checking JavaScript function is called, which performs a file name error check, a filename validity check, and a range check to assure that no internal limits of the build engine are being broken. If the error checking tests are successful another JavaScript function is called to update the panel's database, and set the necessary status variables.

The panel's JavaScript then calls the appropriate build engine method, passing the necessary arguments, including the current internal web page number, the internal number to



US 6,546,397 B1

29

be assigned to the image object, the object type, and the current image style at 96. The build engine then updates its internal database and sets the necessary status variables. It also changes the mouse cursor shape to that of an "Image Create" symbol. In one implementation, the mouse cursor is shaped like an arrow. The build engine detects a mouse click event through its "mouseDown" Method at 97. This method reports to the build engine the exact horizontal and vertical coordinates of the arrow mouse cursor at the time the mouse button was pressed, and places these values into its internal database. The polling process is also handled, as discussed in FIG. 9. The build engine then asserts the necessary security permission for reading from the local disk, if necessary, and attempts to create the necessary image object at the current mouse coordinates at 98, while checking for any exception conditions as described in FIG. 10. If no unrecoverable exceptions are reported, the internal database is updated and the draw system is called.

The image processing technology supports direct web page image object interactions at 99, utilizing the communication technology described in FIG. 6. The build engine first processes the mouse event as described in FIG. 7, and places the appropriate values into a poll enabled JAVA method as described in FIG. 9. There are two types of direct web page image object interactions. The first occurs by simply selecting the image object with a single mouse click. A red selection rectangle is drawn around the image, as are eight attachment points. When the user has pressed the mouse cursor, the mouse cursor's shape changes to that of an anchor, which is a symbol that can be used when dragging or moving an object. The mouse's location will jump to the origin for the image. In an alternative implementation, the anchor can be defined by the mouse location at the time of the mouse drag operation. In either case, while the mouse is being dragged, the build engine updates its internal database. The build engine also updates its poll-enabled methods for communication with the interface's polling technology at 100. The JavaScript poller reads these values, updates the panel JavaScript database, and updates the panel's interface objects. In a similar way, placing the mouse cursor over an attachment point and dragging will result in an image resizing operation. Screen shots FIG. 50 through FIG. 52 show a visualization of an image dragging operation. Screen shot FIG. 50 shows the cursor over an unselected image. Screen shot FIG. 51 shows the screen state after the left mouse button has been pressed. Notice that the image is now selected and the cursor shape has changed to the drag state. Screen shot FIG. 52 shows the screen state after the mouse has been dragged to the northwest. Notice that the image stayed selected and moved with the mouse. Screen shots FIG. 53 and FIG. 54 show a visualization of an image resizing operation for a normal image. Notice that all eight attachment points are drawn and active for the selected image. Screen shot FIG. 53 shows the cursor over the upper left attachment point. Notice that the cursor shape has changed to a northwest to southeast resize cursor shape. Screen shot FIG. 54 shows the result after the left mouse button has been pressed over the upper left attachment point and dragged to the northwest. Notice that the image's upper left corner is still under the cursor, the image has resized, and the cursor shape remained unchanged. For image resizing operations with the mouse over and mouse down objects, only the east, southeast, and south attachment points are drawn and active.

The second type of direct web page image object interaction occurs when the panel's JavaScript code detects that the user has selected an image object interaction feature

30

from the panel's "Image" menu. The appropriate JavaScript function is called, which sets the necessary status variables, and then calls the appropriate JAVA method, passing the necessary values as arguments. The JAVA method then sets its necessary status variables, changes the mouse cursor shape as appropriate, depending upon the type of direct image operation, and awaits a direct mouse operation on the image object. Image rotation is an example of this type of direct image interaction. In one implementation, direct image object rotation is realized by utilizing the image rotation technology described in association with FIG. 33 below. Screen shots FIG. 55 and FIG. 56 show a visualization of an image rotation for a normal image. Screen shot FIG. 55 shows the user selecting the rotate command from the "Image" menu. Immediately the cursor's shape changes to the rotate (a dual left/right arrow) cursor, and the selected image's attachment points disappear. Placing the cursor on the image and dragging will cause the image to rotate on an east/west and/or north south axis. Screen shot FIG. 56 shows the result after the mouse was dragged on an east/west plane.

Image object interactions are invoked by selecting from the JavaScript panel, selecting from a JavaScript pop-up window, and by selecting from a JAVA window object at 101, as described in FIG. 6. The initial values in the pop-up window are set from JavaScript's database. After any user interaction, JavaScript's database is updated and the appropriate method in the build engine is called with the necessary settings. The build engine, after updating its internal database, calls the appropriate image processing method. The image processing routine then calls the required image filter(s), which then perform the necessary processing on the image bitmap at 102.

An image filter is a body of code, usually consisting of one or more digital image processing algorithms, which operate on an image bitmap, and create a transformed image bitmap. An image observer can be invoked by the image filter, which then reports when the image bitmap processing has been completed. An image observer is an independent process that monitors a particular image processing event, such as the execution of an image filter or the reading in of an image file, and reports the status of that process when queried. When the image observer reports a successful completion, the image filter can call the build engine's draw system to display the transformed image bitmap. This interaction between the build engine's image processing method, the image filter(s), the image observer, and the draw system can occur many times, depending upon the image processing operation chosen. Image animations and image transformations, which are technologies that rely heavily on image filters, and the image observer are discussed in greater detail below in association with FIG. 16 and FIG. 17.

FIG. 13 shows a process for implementing text button, image and paragraph style settings (20 of FIG. 3). The initial values for all the settings inside a parent pop-up window and associated child pop-up windows, for a particular style, can be set from the JavaScript database at 103. The settings can include: image object styles, text button object styles and paragraph object styles.

The following settings can be initialized and changed for image object styles. The following settings are initialized for all image object states (Normal, mouse Over, mouse Down) and can be changed:

- (1) resize factor.
- (2) rotation factor.
- (3) main animation type, speed, number of animation steps (resolution) and number of cycles.
- (4) image processing factors. (brightness, contrast, etc.)

## US 6,546,397 B1

31

- (5) 3d effects and their color values.
- (6) web page centering attribute.
- (7) web page scaling attribute.
- b) The following actions are initialized and can be changed.
  - (1) sound effects and audio channels.
  - (2) video files and video channels
  - (3) text button and image pop ups and their attributes (See 1.a above and 2.a below.)
  - (4) click events.
- c) The following transformation settings are initialized and can be changed.
  - (1) the initial delay
  - (2) up to three transformations can be defined with the following settings:
    - (a) which image states should the transformation be from and into.
    - (b) the speed of the transformation.
    - (c) any delay before the next transformation.
  - (3) whether the transformation(s) should occur simultaneously with the enter and exit time line animation or after the enter and before the exit animations.
- d) The following time line settings are initialized and can be changed.
  - (1) the initial delay before the image object's appearance.
  - (2) the enter animation type, speed, and animation resolution.
  - (3) the delay after the enter animation and the main animation.
  - (4) the exit animation type, speed, and animation resolution.
  - (5) the initial delay, after the entrance of the parent object, before the child text button and image object's appearance(s).
  - (6) the child object(s) enter animation type, speed, and animation resolution.
  - (7) the delay after the child object(s) enter animation.
  - (8) the child object(s) exit animation type, speed, and animation resolution. The following settings can be initialized and changed for text button object styles.
- e) The following attributes are initialized for all text button object states (normal, mouse over, mouse down) and can be changed:
  - (1) all font specifications.
  - (2) vertical state.
  - (3) all color specifications.
  - (4) 3d effects and their color values.
  - (5) web page centering attribute.
  - (6) font processing attributes (available in java 2)
  - (7) scale, shear, and rotate (available in java 2)
- f) The following actions are initialized and can be changed.
  - (1) sound effects and audio channels.
  - (2) video files and video channels
  - (3) text button and image pop ups
  - (4) click events.
- g) The following transformation settings are initialized and can be changed.
  - (1) the initial delay
  - (2) up to three transformations can be defined with the following settings:
    - (a) which image states should the transformation be from and into.
    - (b) the delay before the next transformation.
  - (3) whether the transformation(s) should occur simultaneously with the enter and exit time line animation or after the enter and before the exit animations.

32

- h) The time line settings are the same as those defined for image objects. They also are initialized and can be changed.

The following settings can be initialized and changed for paragraph styles.

The following attributes are and can be changed:

- i) all font specifications.
- j) all color specifications.
- k) 3d effects and their color values.
- l) web page centering attribute.
- m) the look of hot links, including the text and background colors when the link is active and when the mouse is over the link.

The reference to JAVA 2 under text button object styles refer to the most recent version of JAVA released by Sun Microsystems. This version supports a far more robust two-dimensional processing capability than JAVA 1.6, including significant font processing capabilities and the scaling, shearing, and rotation of objects. Currently, most conventional browsers only support JAVA 1.6. Provisions are made in the invention so that as the then popular browsers support more robust versions of programming languages, those new capabilities can be employed to further enhance the capability of the invention.

Referring again to FIG. 13, upon detecting the completion of editing an image, text button or paragraph style, the panel's JavaScript calls a build engine method and passes the required values. The build engine updates its internal database and sets any necessary feature flags at 104. When an image, text button or paragraph object is created, all the style settings for the currently selected style are applied by the build engine as part of the definition for the newly created object at 105.

If a style is changed, all objects on all internal web pages that are utilizing that style are candidates for being changed to those new values at 106. Flags are kept for every possible style setting for each object. If a given object is edited through the text button, image, or interaction menus or other interface objects of the panel 400, the flags are set for any setting that are changed. If that style is subsequently changed, only those settings that have not had their flags set will be changed for any given object.

FIG. 14 describes the video and audio file and video and audio channel processing techniques employed by the build engine (21 of FIG. 3). A user can select a video or audio special effect (i.e. user input is provided at 107 that indicates a video or audio special effect). The method for activating a video file or video channel is defined in the text button and image object "mouse over" interactive pop-up windows described later at FIG. 16. Methods for defining a video object as a pop-up, or a frozen object, are described with reference to the text button and image object "mouse down" interactive pop-up window also described at FIG. 16. Audio files and audio channels can be defined in both the "mouse over" and "mouse down" interactive pop-up windows also described at FIG. 16. The pop-up or a frozen object settings for audio are also set in the object "mouse down" interactive pop-up windows discussed therein.

As before, the panel JavaScript code initializes any pop-up windows (where the initial values are set from the JavaScript database), captures a file or channel name (from the user input) and performs file and validity error checking upon detecting a user completion action at 108. The build engine is then called, receiving the necessary data and status as function call arguments. The build engine determines if the audio and video definition is a file pathname or the URL.

US 6,546,397 B1

33

of a live channel at 109, and thereafter initiates its exception handling. If the video or audio definition is a file, the build engine performs the relevant file exception handling checks, and asserts the necessary security permissions. If there were no errors, or the exception handling error was recoverable, the build engine reads and links the video/audio file to the database, and plays the file for user verification at 110. If the video or audio definition was a channel, the necessary pointers are updated in the database, and methods are assigned for efficient transmission, at run time by the run time engine, at 111. The ability of the run time engine to play multiple synchronized audio and video files and channels simultaneously will be described at FIGS. 31-35.

FIG. 15 describes the frames, tables, forms and draw objects technologies employed by the build engine (22 of FIG. 3) in one implementation of the invention. When the panel JavaScript code detects a user action to create a "frame", "table", "form" or "draw object" from an appropriate panel interface object, it draws and initializes the appropriate pop-up window at 112. Upon detecting a user completion action by the pop-up window's JavaScript code, a panel JavaScript function is called to perform the necessary error checking and updating of the panel's database. Panel JavaScript thereafter calls the appropriate build engine method(s) passing the necessary data and status values as function call arguments at 113.

The build engine updates its internal database, sets the necessary status values, and initializes, as necessary, appropriate methods for run time processing. In one implementation, the build engine includes definitions to map a given object into a relational database. Also available are a full array of database operations. Support for popular databases (such as Oracle, Informix, Sybase and DB2) are available on a real time interactive basis.

The run generation technologies, as described later in FIGS. 24-27, are also implemented for a given frames, table, form and draw object at 114. The run time technologies, as described later in FIGS. 28-36, are also implemented for a given frame, table, form and draw object at 115. FIG. 16 describes the user interaction settings and technology employed by the build engine (24 of FIG. 3). Depending upon the type of object currently selected at 116 (if no object is selected no user interaction choices will be available) the panel JavaScript Code draws an appropriate pop-up window. If the selected object was a text button object at 117, or an image object at 119, both "mouse over" and "mouse down" choices will be available from the panel's "Interactions" menu. If the selected object is a paragraph, user interaction definitions can be activated by a double click or a mouse drag event being detected by the build engine at 118.

More specifically, appropriate values are set in a poll-enabled JAVA routine. The JavaScript poller reads the values, and draws the appropriate panel menu choices. The "Text Style", "Hot Link", "Preferences" and "Format" pop-up windows can be chosen. If the hot link choice under the panel's "Text" menu is selected and executed, the hot link definition for internal or external web pages is captured by an appropriate JavaScript function and file pathname error and validity checking is performed. If either the "Text Style", "Hot Link", "Preferences" or "Format" choices under the panel's "Text" menu are selected, the panel's JavaScript draws the appropriate pop-up window. Upon detecting a user completion event, the panel's JavaScript reads the values in the pop-up window and passes the font specification parameters to an appropriate build engine method as function call parameters. The build engine then processes this data, calls

34

a reformat method, updates its internal database, and sets the necessary four-dimensional variables for communication with the draw system.

The normal and "mouse over" foreground and background colors for the hot link, which were defined in a link look pop-up window (available under the "Text" menu of the panel), are utilized by the build engine to draw the hot link. The build engine performs the necessary exception handling, and then updates its internal database.

Based on the panel's JavaScript Code detecting whether the user chose the "mouse over" or "mouse down" choice under the "Interactions" menu at 120, as well as based on whether an image or text button object is currently selected, the panel's JavaScript code draws the appropriate pop-up window. Initial values for the pop-up windows are set from the panel's database at 121 and 122. In one implementation, the following user interaction's for the "mouse over" and "mouse down" states for text button and image objects are supported:

- 1: 3D Frame, in a specified color, and selected for a specified 3D appearance, can be defined for text button and image object's "mouse over" and "mouse down" states, as well as for their text, image and video pop-ups.
- 2: The font typeface, font style, font size, font effect(s), text color, and text background color can be defined for a text button object's "mouse over" and "mouse down" states, as well as for the text pop-ups associated from both text button and image objects.
- 3: Text, image, and video pop-ups can be defined for the text button and image object's "mouse over" state.
- 4: A sound track (file) can be defined for the text button and image object's "mouse over" state with the following choices:
  - a. play once when a "mouse over" event occurs.
  - b. play until a click event while on the object.
  - c. play until the mouse moves off the object.
- 5: A sound track (file) can be defined for the text button and image object's "mouse down" state with the following choices:
  - a. play once when a mouse click event occurs when over the object.
  - b. play until a second click event while on the object.
  - c. play until the mouse moves off the object.
- 6: Both video and sounds can be defined as channels as well as files.
- 7: The text, image, and video pop-ups can be frozen (i.e. not disappear when the mouse moves off the object after a mouse click event, for both text button and image objects).
- 8: Text button and image objects can have one of the following click events defined:
  - a. go to a specific internal web page.
  - b. go to the next internal web page.
  - c. return to the parent (calling) web page.
  - d. go to an external web age. That web page will replace the current web page.
  - e. go to an external web page. That web page will be launched into a new window so that both web pages will be visible and accessible.

After a user completion action is detected, the panel JavaScript code performs the necessary file error and validity checking, updates its database and sets necessary status values, and then calls the appropriate build engine method, passing the necessary data values and status as function call arguments at 123. The build engine updates its internal



US 6,546,397 B1

35

database, sets the necessary status variables, then draws the appropriate "mouse over" or mouse down" text button or image object states. The build engine also plays the sound or video file for user verification. The run time technology behind the user interactions will be described in greater detail in association with FIG. 36.

FIG. 17 describes the image and text button object animation settings and technology employed by the build engine (25 of FIG. 3). The panel's JavaScript code determines which type of object, and which object number, from the currently selected object, as reported by the poller at 124. When the panel's JavaScript detects a user selection of "Define Image" or "Animate" from the panel's "Image" menu, or a user selection of "Define Button" or "Animate" from the panel's "Button" menu, it draws the appropriate pop-up window and initializes the pop-up window's values from its database at 125 and 126. Screen shot FIG. 57 shows a visualization of one implementation of the "Text Button Animation Specifications" pop-up window and the animation settings available to the user. Screen shot FIG. 58 shows a visualization of one implementation of the "image animation specifications" pop-up window and the animation settings available to the user.

When a user completion event is detected, the panel's JavaScript code captures the values from the pop-up window for the animation type, speed, resolution, and number of animation cycles at 125 and 126, respectively, and updates its database at 127. The panel's JavaScript then calls the appropriate build engine method, passing the necessary data and status values as function call arguments. The build engine updates its internal database and sets the necessary feature flags (See FIG. 8.) Linkage to the appropriate animation method(s) is also set.

A thread object (a thread is an independent asynchronous program that is multiprogrammed with other threads, are defined and executed by the invention, by a JAVA Virtual Machine and by the browser) is created and executed for user verification at 128. Values are set to integrate the given animation thread with the object time line technology (See FIG. 19). Values are set at 129 so that when the thread object is invoked by the run time engine, the appropriate image filter(s) and animation methods are called. The run time technology behind image and text button object animations is described in greater detail in association with FIG. 31 through FIG. 35.

FIG. 18 describes the transformation settings and technology utilized by the build engine (26 of FIG. 3). A transformation is defined as the changing of an object from one state to another based on a timer control, subject to user settings. In one implementation, the available states for text button and image objects are their "normal", "mouse over", "mouse down" and "pop-up" definitions. For text button objects, a transformation is implemented as the instantaneous drawing of one object state while erasing the previous object state. For images, a transformation is the gradual fading out of the previous object state, while, simultaneously, fading into the next object state.

Prior to any user menu selection, the panel's JavaScript code already knows the status of any selected object through the poller mechanism (124 of FIG. 17). This includes what type of object and the object's internal identifying number. When the panel's JavaScript detects a user selection of "Transform" from the panel's "Interactions" menu, it draws an appropriate pop-up window and initializes the pop-up window's values from its database at 130. Screen shot FIG. 59 shows a visualization of one implementation of a "define the transformation for the text button object" pop-up win-

36

dow and the transformation settings available to the user. Screen shot FIG. 60 shows a visualization of one implementation of a "define the transformation for the image object" pop-up window and the transformation settings available to the user. When a user completion event is detected, the panel's JavaScript Code captures the values from the pop-up window based on the object type.

In one implementation, the following settings for text button objects can be specified:

1. The initial delay.
2. Up to three transformations can be defined with the following settings:
  - a. Which image states should the transformation be from and into.
  - b. The delay before the next transformation.
3. Whether the transformation(s) should occur simultaneously with the enter and exit time line animation or after the enter and before the exit animations.

In one implementation, the following settings for image objects can be specified:

1. The initial delay.
2. Up to three transformations can be defined with the following settings:
  - a. Which image states should the transformation be from and into.
  - b. The speed of the transformation.
  - c. The resolution of the transformation.
  - d. Any delay before the next transformation.
3. Whether the transformation(s) should occur simultaneously with the enter and exit time line animation or after the enter and before the exit animations.

The panel's JavaScript updates its database at 131. The panel's JavaScript then calls the appropriate build engine method, passing the necessary data and status values as function call arguments. The build engine updates its internal database and sets the necessary feature flags (See FIG. 8.) Linkage to the appropriate transformation method(s) is also set.

A thread object is created and executed for user verification at 132. Values are set to integrate this transformation thread with the object time line technology (See FIG. 19). Values are set at 133 so that when the run time engine invokes the thread object, the appropriate image filter(s) and transformation methods are called. The run time technology behind image and text button object transformations is described in greater detail below in association with FIG. 31 through FIG. 35. FIG. 19 describes the text button and image time lines and technology utilized by the build engine (27 of FIG. 3). A time line is an independent asynchronous process that defines the existence of a given text button or image object. An object's time line begins at the time a given web page makes its appearance, either through an immediate draw or through a transition animation. In one implementation, an object time line can be created as an instance of a class, which has a threadable interface. This instance has its own data structures, which define the animations, and transitions associated with the time line definition. An image or text button object time line can spawn child time lines, at a designated moment. A complete description of time line technology, and how they integrate the animation and transformation technologies, will be described below in association with FIG. 31 through FIG. 35.

The build process begins the time line definition process by having the panel's JavaScript determine what is the currently selected object, utilizing the polling technology at

US 6,546,397 B1

37

134. That is, values for the object's appearance time, animation type, speed and resolution are captured. When the panel's JavaScript detects a user selection of "time line" from the panel's "Interactions" menu, it draws the appropriate pop-up window and initializes the pop-up window's values from its database. Screen shot FIG. 61 shows a visualization of a collage of one implementation of a "define the time line for the text button object" tabbed pop-up window and the time line settings available to the user under each tab. Screen shot FIG. 62 shows a visualization of a collage of one implementation of a "define the time line for the image object" pop-up window and the time line settings available to the user under each tab. When a user completion event is detected, the panel's JavaScript captures the values from the pop-up window based on the object type. The currently available settings, for both text button and image objects, are:

- 1: The initial delay before the image object's appearance.
- 2: The enter animation type, speed, and animation resolution.
- 3: The delay after the enter animation and the main animation.
- 4: The exit animation type, speed, and animation resolution.
- 5: The initial delay, after the entrance of the parent object, before the child text button and image object's appearance(s).
- 6: The child object(s) enter animation type, speed, and animation resolution.
- 7: The delay after the child object(s) enter animation.
- 8: The child object(s) exit animation type, speed, and animation resolution.

The panel's JavaScript updates its database at 135. The panel's JavaScript then calls the appropriate build engine method, passing the necessary data and status values as function call arguments. The build engine updates its internal database and sets the necessary feature flags (See FIG. 8.) A build engine method then processes all the data related to this object. The object's animation settings, if any, are integrated into the timeline at 136. The object's transformation settings, if any, are also integrated into the timeline. If an image object, any transformation animation may be executed simultaneously with the appearance and/or exit animations, depending upon the settings. Finally, a multi-level object thread definition is created and executed for user verification. Values are set at 137 so that when the run time engine invokes the thread object, the appropriate image filter(s), animation methods, and transformation methods are called.

FIG. 20 describes the web page transition animations, time line settings and technology utilized by the build engine (28 of FIG. 3). When the panel's JavaScript detects a user selection of "Define" from the panel's "Webpage" menu, it draws the appropriate pop-up window and initializes the pop-up window's values for the current web page from its database at 138. Screen shot FIG. 63 shows a visualization of one implementation of the "define the current web page settings" pop-up window and the web page settings available to the user. In the implementation shown, the choices supported include:

- 1: The web page delay time (which is the delay, after the completion of the last object time line, to the loading of the next web Page).
- 2: The transition animation, which can include a random animation choice. This is the animation applied to the

38

web page when it is loaded and to the previous web page as it departs.

- 3: The number of animation frames per second, which effectively is the resolution of the transition animation.
- 4: The number of animation frames, which effectively defines the time expected for the transition animation to complete.
- 5: The web page's background color. This setting will override the generic setting for the web site, defined in FIG. 21a.
- 6: A web page boarder. This boarder, if selected, will also override the setting for the web site, defined in FIG. 21a. The boarder can be drawn with a 3D effect, taking the background color, and applying a transformation so that, to the human eye, a lighter and darker shade of that color will be drawn appropriately to create a 3D effect.
- 7: The web page's background pattern. This setting will override the generic setting for the web site, defined in FIG. 21a.

The panel's JavaScript updates its database at 139. The panel's JavaScript code then calls the appropriate build engine method, passing the necessary data and status values as function call arguments. The build engine updates its internal database and sets the necessary feature flags (See FIG. 8.). The web page time line is synchronized with its object time lines by an appropriate build engine method at 140. The web page's appearance delay and transition settings are integrated into the web page time line. Thereafter, a single-level object thread definition is created. Values are set at 141 so that when the thread object is invoked by the run time engine, the appropriate animation methods and object time line threads are called. Again, the run time technology behind web page transition animations and web page time lines is described in greater detail below in association with FIG. 31 through FIG. 35.

FIG. 21a describes the file operations supported by the build engine (29a of FIG. 3). In one implementation, the file operations supported include:

- 1: "Save" at 142 or "Save As" at 143. If the selection from the panel's "File" menu is to "Save" as a web page, the current browser screen height percentage value is sent to the build engine. The build engine updates its internal database and the build process is completed. Thereafter, the run generation process is executed. (See FIG. 24 through FIG. 27.) If the selection is to "Save As" a template for the run generation process is also executed but the generated files are placed in the template directory. If the selection is to save as a banner or custom application, those absolute screen dimensions are sent to the build engine and its internal database is updated and the run generation process is executed.
- 2: "New" at 144. A test is made by the panel's JavaScript code to see if any user input has been processed and not saved. If so, the user is asked whether this data should be saved. If so, and if the selection is to "Save" as a web page, the build process is completed and the run generation process is executed as described above. If the selection is to "Save" as a template the run generation process is executed but the generated files are placed in the template directory as described above. The panel's JavaScript code then reinitializes its database and calls a build engine method that reinitializes the build engine database.
- 3: "Close" at 145. A test is made by the panel's JavaScript to see if any user input has been processed and not

US 6,546,397 B1

39

saved. If so, the user is asked whether this data should be saved. If so, and if the selection is to "Save As" a web page, the build process is completed and the run generation process is executed. If the selection is to "Save As" a template the run generation process is executed but the generated files are placed in the template directory. The panel's JavaScript then terminates the build process.

- 4: "Open" at 146. A test is made by the panel's JavaScript to see if any user input has been processed and not saved. If so, the user is asked whether this data should be saved. If so, and if the selection is to "Save As" a web Page, the build process is completed and the run generation process is executed. If the selection is to "Save As" a template the run generation process is executed but the generated files are placed in the template directory. The panel then initiates the dynamic web page resizing technology as described in FIG. 22 below for the open re-initialization mode.
- 5: "Apply" at 147. A template is applied to the existing web site that is being processed by the build engine. The web page and style record definitions of the template replace those of the existing web site. The web page objects of the template are added to the web page objects of the existing web site.
- 6: "Web Site" at 148. The web designer can define settings that will be applied to all web pages in the web site. In one implementation, the web site applications supported include:
  - a: web page. The web page height can be set, as a percentage, larger than the browser window for long vertically scrolled web pages.
  - b: Standard banner sizes.
  - c: Custom. (The user can define any arbitrary web page size and resolution) Screen shot FIG. 63 shows the generic web site setting choices presented to the user in one implementation of the invention.

FIG. 21b describes the view operations supported by the build engine (29a of FIG. 3). In one implementation, the file operations supported include:

- 1: "Normal" at 149a. This is the default file mode in which the interface and the build engine are processing user input as described in FIG. 5 through FIG. 23 above.
- 2: "Preview" at 149b. The build engine runs the web site off its internal database. The web site will perform in an identical manner as if it had gone through the entire run generation and run time process, but it is being executed on the web designer's computer.
- 3: "Play" at 149c. The build engine runs the web site off an external database in a separate browser window. The web site will perform in an identical manner as if it had gone through the entire run generation and run time process, but it is being executed on the web designer's computer.
- 4: "Zoom" at 149d. The dynamic web page resizing technology (see FIG. 22 below) is first executed. When the engine is fully reinitialized, and the engine has gone to the current web page, the page and all its objects are drawn to the scale as defined by the zoom level. All object coordinates and sizes are automatically scaled appropriately because they are always defined with virtual screen values, even when the web page is being draw in the "normal" view.

FIG. 22 describes the dynamic web page resizing technology supported by the build engine. If a user selection of

40

the "Open" command from the "File" menu is detected by the panel at 500, the panel calls an engine method to read selected contents from that web site's external database file.

In one implementation of the invention at 506, the engine reads the web page width and length fields, as well as the background color or background image definition for the first web page of the Web Site. The engine then creates a build engine HTML definition file based on these specifications, and writes this file either to the local disk or the server, depending upon the origination of the build tool.

At 502, if a user completion event occurs inside the web site JavaScript pop-up window, which had been activated when the user selected the "Web Site" command from the "File" menu, the panel determines if the web site page size has been changed. If so, the panel calls an engine method for processing.

Similarly, at 504, if a user selection of a "Zoom" command from the "View" menu is detected by the panel at 504, the panel also calls an engine for processing.

In both the cases at 502 or 504, in one implementation of the invention, the engine writes out a checkpoint record at 508 that is similar to that of a "Websitename".dta "database file (See FIG. 24). But is given the temporary checkpoint Websitename. The engine then creates a build engine HTML definition file based on these specifications, and writes this file either to the local disk or the server, depending upon the origination of the build tool.

In one implementation of the invention at 510 the engine terminates itself, by stopping all of its threads. Meanwhile the interface writes out four cookies onto the local disk which define the following:

- 1: The re-initialization mode. (Either Open or Checkpoint).
- 2: The current web page number when the resizing event occurred.
- 3: The Web Site Name. (The checkpoint name if in checkpoint mode)
- 4: The zoom level.

The interface then terminates itself by executing the JavaScript "parent.location.href" command, which causes the build engine parent HTML frame file (PFF) to be reloaded (See FIG. 5).

In one implementation of the invention at 512 the re-initialization process begins. The PFF cause both the panel and the build engine to be reloaded and activated. The panel then reads the mode cookie. If the mode is either open or checkpoint, the interface reads the web site name, page number and zoom level cookies, then resets the mode cookie to the initialize state for subsequent operations. The interface then calls an engine method to read the external database, and then to return the necessary values from that database in order to update the interface's database. Finally the engine calls two engine methods in order for the engine to go to the correct current web page and draw that page at the now current zoom level. Normal processing can then resume.

Run Generation Process

FIG. 24 through FIG. 27 describe the run generation process. This process starts when the build process detects that the user is finished defining the web site (user has saved the web site and invokes the run generation process), and concludes with the actual uploading of all the necessary web site run time files to the user's server.

FIG. 24 describes the techniques employed by the build engine for the creation of the external database, and the security and optimization techniques that support this process (30 of FIG. 4).

When the panel's JavaScript Code detects a user selection of "Save" or "Save As" from the panel's "File" menu, it



US 6,546,397 B1

41

draws the appropriate pop-up window and initializes the pop-up window's values for the current web page size as had been defined at FIG. 5 and passed to the build engine. The panel's JavaScript in the "save the web page/template" pop-up window detects a user completion event at 150 (i.e., the designation of a user's web site name followed by the enter key), and calls the appropriate panel JavaScript function. More specifically, after completing the appropriate validity checks, the function calls the appropriate JAVA build engine method, passing as a function call argument the user defined "Websitename". The build engine method checks for the existence of a "Websitename".dta file, and, if so, posts that result into a poll-enabled method return value. The poller checks that value, and if set to true, calls a JavaScript function which draws a pop-up window asking the user to confirm whether the existing web site definition should be overwritten or not. This JavaScript function also calls an appropriate build engine method to reset the return value to false in order to be initialized for the next possible "Save" operation.

Once this verification process is completed the build engine begins the external database creation process at 151, which will vary depending upon the security manager of a given browser at 152. See FIG. 5 for a detailed description of the browser security manager alternatives. If the browser's security manager allows for local disk file creation, the build engine calls a method, which asserts the necessary security policy permissions to create and write a file. If not, the build engine calls the necessary method to create and write a file on the user's server.

The external database contains, as its first record, a "Header" record, which contains can include the following information:

- 1: A file format version number, used for upgrading database in future releases.
- 2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user at FIG. 5.
- 3: Whether the application is a web site.
- 4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.
- 5: Web page and styles high watermarks.
- 6: The Websitename.

The header records are written at step 153.

During the build process, as new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, at 154, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the external database as described at 156, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

42

The settings for all paragraph, text button and image styles are then written as a style record at 155 based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist. The font and color objects are serialized as is discussed in greater detail below (See 159 below).

The body of the external database is then written at 156. All Boolean values are written inside a four-dimensional loop at 157. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects (i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of line segments per paragraph line and contains the values used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment.).

All integer values are written inside a four-dimensional loop at 158. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

Multimedia objects are written inside a two-dimensional loop at 159. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other objects on a web page.

String records are written inside a four-dimensional loop at 160. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The third loop contains values for paragraph line related data, and the innermost loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized.

US 6,546,397 B1

43

Single and double floating-point, and long integer records are written inside a two-dimensional loop at 161. The outer loop may be empty. The inner loop contains mathematical values required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

FIG. 25 describes the techniques used to create a customized and optimized run time engine by the build engine (31 of FIG. 4). A versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted at 162. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation at 163.

All external image, video and audio files are resolved at 164. The external references can be copied to designated directories at 164, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries from Sun, Microsoft and Netscape are either installed on the local system (See FIG. 5) or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code at 165. Finally, the run time engine for the web site is created at 166. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file (See FIG. 27).

FIG. 26 shows the techniques used to create the HTML Shell File (HSF) (32 of FIG. 4). The first step of the process at 167 is to determine whether the dynamic web page and object resizing is desired by testing the application setting, set by the user at FIG. 21a, or possibly reset at FIG. 24. If the application was a web page, and thus requiring dynamic web page and object resizing, virtual screen resolution settings, calculated at FIG. 24 at 153, are placed in an appropriate HTML compliant string at 168. If the application is a banner or other customized application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values at 169.

An analysis is made for the background definition for the first internal web page at 170. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the browser. More specifically, if the application required dynamic web page and object resizing (See 167) then JavaScript and HTML compliant strings are generated at 171 so that, when interpreted by the browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated at 172 in order to enable JavaScript to SRS applet communication. In

44

one implementation, the code is generated by performing the following functions:

- 1: Determine the current browser type.
- 2: Load the SRS from either a JAR or CAB File, based on browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.
- 5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated at 173 that perform the following steps at the time the HSF is initialized by the browser:

- 1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.
- 2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the External Database created at FIG. 24.
- 3: Generate an HTML complaint string, dependent upon the type of browser, which causes the current browser to load either the JAR or the CAB File(s).
- 4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

At 174, writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "Website-name".html file is created.

FIG. 27 describes the processes for creating the CAB and JAR Files (33a of FIG. 4). The image objects, if any, which were defined on the first internal web page are analyzed at 175. If they are set to draw immediately upon the loading of the first web page, then they are flagged for compression and inclusion in the CAB and JAR Files. The feature flags are analyzed at 176 to determine which JAVA classes have been compiled (See FIG. 25). These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created at 177 that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "Website-name".class, customized run time engine file, and the "Website-name".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "Website-name".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "Website-name".bat and "Website-namelib".bat files are written at 178. The "Website-name".bat and "Website-namelib".bat files are then executed under DOS, creating compressed "Website-name".cab and "Website-namelib".cab files and compressed "Website-name".jar and "Website-namelib".jar files at 179. The HTML Shell File and the JAR and CAB files are then, either as an automatic



US 6,546,397 B1

45

process, or manually, uploaded to the user's web site. This completes the run generation processes.

#### Run Time Process

The run time process is shown in FIG. 28 through FIG. 36.

FIG. 28 shows the web page size generation technology utilized by the run time engine. A web surfer points a browser at the HTML shell file (HSF) at 180. The browser begins to interpret the HTML and JavaScript code in the HSF that was created (See FIG. 26). The browser draws either the background color or background image pattern, as defined by the HTML complaint code (See FIG. 26) at 170. The browser then executes the HSF's JavaScript initialization code, which "sniffs" the current browser at 181 to determine its type, and then generates the appropriate HTML code for that particular browser to interpret. This code defines whether the executable files and database will be extracted from inside a compressed CAB file or a compressed JAR file and its location.

Based on the user application (defined at FIG. 21a, or possibly reset at FIG. 24), the HSF at 182 will then execute an appropriate JavaScript function (as created in FIG. 26 at 167). If the application required dynamic resizing of the web page's dimensions, JavaScript code is called which generates HTML code using the JavaScript "document.write" function, which causes the SRS applet to be immediately executed by the browser at 183. The JavaScript code then goes into a timer loop, checking on when the SRS applet is alive before initiating any communication. After detecting that the SRS has been initialized, a JavaScript function calls the appropriate SRS applet methods at 185, which return the width and height, in pixels, of the current browser window. JavaScript Code is then called which converts the screen resolution independent window width and height values into absolute pixel values. A JavaScript function is then called which use the JavaScript "document.write" function to generate HTML code that define the run time engine specifications, etc. (see FIG. 26) and cause the browser to immediately execute the run time engine. If the application had not required dynamic resizing of the web page's dimensions, then a JavaScript function is called which generates HTML code using the JavaScript "document.write" function that defines the fixed dimensions for the web page size and cause the browser to immediately execute the run time engine at 184.

FIG. 29 shows the techniques employed by the run time engine to read the external database and to generate the necessary web page objects (35 of FIG. 4). The run time engine reads a "PARAM" value at 186, from HTML Code that was generated above (see FIG. 26), which points to the "Websitename".dta external database that is compressed into the JAR or CAB File (that was loaded and accessed in FIG. 28). The run time engine then initiates the read operation. In one implementation, the read technique is always non-privileged. If permitted by the current browser as a non-privileged operation, the "Websitename".dta file will be extracted and read from the CAB/JAR file residing in temporary local storage. If not, the run time engine will read the "Websitename".dta file directly from the server. The header record is read at 187. Any objects, such as fonts and colors, are cast into their original form. The high watermark values, as they are encountered in the header and in the body of the database, are immediately used for setting the limits for the subsequent multilevel read loops for reading in the style record and the web page(s) and object(s) definitions. The virtual screen resolution values are read for the subsequent dynamic resizing of the web page objects.

The style record is read based on its high watermarks, and processed at 188. The definitions for all paragraph, text

46

button, image or other styles are read and stored for subsequent initialization and processing of all paragraph, text button, image or other objects. The data representing the values for the first web page and all its objects is read at 189. The Boolean, integer, string and floating point fields for the first web page are initialized. The serialized multimedia objects for the first web page are read and cast into their final form. (See FIG. 24)

If external files, such as image, audio and video files, must be read as part of the first web page's generation, exception handling routines are executed at 190, as necessary, in the event of any processing errors. In one implementation, error recovery at this stage places the highest priority on a graceful operation cancellation, rather than a web page crash. In the worst case, a particular image, sound or video file may not be available to the web surfer. All other aspects of the web page will likely be available even in this error scenario.

Process step 191 is executed simultaneously with the generation of all the other web pages at 192 by means of multiprogramming utilizing thread technology. Thus the first web page will be drawn and active for user viewing and user interaction long before the data for all the other web pages have been read, processed, and initialized. The data representing the values for the subsequent web pages and all their objects are read at 192. The Boolean, integer, string and floating point fields for these web pages are initialized. The serialized multimedia objects for these web pages are read and cast into their final form. (See FIG. 24)

FIG. 30 shows the scaling techniques employed by the run time engine for web page generation (36 of FIG. 4). The first step in the scaling process is to calculate the coordinates that define the origin for the placement of each object for a given web page. (This is usually the upper left corner of the object, defined in actual screen pixels.) A test is made at 193 to determine if the centering attribute is set for the object. If not, the left and top coordinates are converted from the virtual screen values to the local screen values, based on the local screen window resolution at 194. In one implementation, multiplying the virtual coordinate by the local screen window resolution and dividing by the virtual screen resolution determine the conversion.

If the centering attribute is on, then a calculation for the object's width is performed. See processes 197, 198, and 199 below for a description of this calculation. Based on this calculated width, and based on the local screen window resolution, the left coordinate is calculated at 195. One algorithm that can be used is to subtract the screen width, as calculated in 197-199 below, from the local screen window resolution, and divide that result by 2. The top coordinate is calculated the same as in process 194 above.

Based on the object type, determined at 196, a different scaling technology is employed. If the object is a text button object at 197, the text button object itself, including its background, is not scaled. The virtual width and the local screen width remain the same. However, if a 3D Frame effect is defined, it is scaled based on the following algorithm: if the text string's orientation is Left to Right, the inner width of the 3D Frame, and its placement relative to the text string, is calculated as the length of the text string, plus 1/5 of an "n" space on each side, plus an additional offset appended to the right of the inner width to compensate for the italic font style, if defined for the font of that text string. The italic offset can be defined as the font size for the text string, divided by 10, plus 1. The inner height of the 3D Frame can be defined as the font height plus 2 pixels. The font height equals the font's leading plus its ascent plus its

US 6,546,397 B1

47

descent specifications. The inner height origin can equal the text string origin. The style of the 3D effect (i.e., either a 3D raised look or a 3D depressed look), plus the inner width and height, is sent to a 3D frame build method for the construction of the 3D frame. The width of the 3D frame in pixels can be calculated as the inner width divided by 10 plus 3.

If the text string's orientation is vertical, the inner width of the 3D Frame is an "m" space. The inner height of the 3D Frame can be calculated as the font height times the number of characters in the text string. Both the left and top placement of the 3D frame can be set to the left and top origin of the text string. The width of the 3D Frame can then be calculated as the inner height divided by 10, plus 3.

If an animation is assigned to the text string, the font size used for the initial calculation of the 3D frame is the same as that used to define the animation's initialization value. If the object is a paragraph at 198, and the scaling attribute is on, the maximum width for the paragraph can be defined by the attached paragraph style (or paragraph override) as a percentage of the screen width. This screen width percentage can be converted into an actual width in pixels, based on the local screen's window resolution. If the current screen resolution is the same as that used by the web designer, then the paragraph line end values (just read from the external database) are used without adjustment, bypassing the entire paragraph reformat process. If the current screen resolution is different than that of the virtual screen resolution, then a very compact method of reformat is called (relative to the build engine reformat methods at FIG. 6 and at FIG. 18), and the text for the paragraph is reformatted based on this width.

The run time engine's reformat technology begins by creating one paragraph line for the entire text string assigned to the paragraph text area. All the style overrides are renumbered sequentially with the style records or the non-marked text strings ignored. A simplified "Overflow" reformat method can be called, which chops up the single paragraph line first into paragraph line segments, where each word is defined as a line segment. Because of the word oriented style override architecture, the style overrides have a one-for-one correspondence with the line segments. Each paragraph line break can be calculated by relying on the simplified word oriented style override technology described above. The paragraph line can be built inside a tight word-by-word loop, with a simple logic check for a style override or hard line break. The paragraph width is then derived as the width of the longest line of the reformatted paragraph, while the paragraph height is defined as the font height times the number of lines. If a 3D frame was defined for the paragraph, it can be scaled based on the following algorithm:

The inner width is defined as the same as that of a text string, but the width of the text string for the longest line is used. The same "n" space and italic offset calculations are used. The inner height is calculated as the font height times the number of lines plus 2 pixels.

If the object is a paragraph, and the scaling attribute is off, then the paragraph is treated the same as a text button object, with the only exceptions that there is no vertical orientation, and the height and width of the 3D frame, if defined, is calculated using the same algorithm as was used for the scaled paragraph above.

If the object is an image at 199, and the scaling attribute is on, the image width can be calculated as the virtual width times the local screen window width divided by the virtual screen width. The image height can be calculated as the virtual height times the local screen widow height divided by the virtual screen height. If the image had been resized or

48

rotated, then the virtual width and height of the image would differ from that of that of the original image. If a 3D frame is defined for the image, it can be scaled based on the following algorithm:

The inner width and the inner height of the 3D frame will coincide exactly with the outer edges of the image, after the image had been scaled. Adding the scaled image height to the scaled image width and dividing the result by 40 and adding 3 can calculate the width in pixels of the 3D frame.

If an animation is assigned to the image, then the animation's initialization values for the image's width and height can be used to calculate and draw the initial 3D frame. The coordinates and sizes for the backgrounds for text button, image and paragraph objects can be calculated using the same algorithms as was employed for the calculation and placement of the inner width and inner height for the 3D frame for each object.

FIG. 31 through FIG. 35 shows the multilevel web page and object thread technology employed by the run time engine. The description includes all the animation technologies, transformation technologies, time line technologies and drawing technologies that support this multilevel architecture.

FIG. 31 describes the initial processes for the invention's multilevel web page and object thread technology employed by the run time engine (37 of FIG. 4). Upon the completion of the processing of all the data definitions for the first internal web page (FIG. 30), the main web page thread is created and executed. This causes the run method for the main run time engine class to be executed simultaneously with the reading, processing, and scaling of the data for the subsequent web pages (See FIG. 29). In addition, the reading of any image files defined for the first web page is also performed simultaneously, under the control of an image observer (See FIG. 12). The main run method enters a web page counter loop at 200, the loop being defined from the first internal web page to the high watermark that was set to the number of existing internal web pages for the web site.

A check is made at 201 to see if the current web page exists. If the web page does not exist, and the current web page number is less than that of the high watermark, then the web page counter is incremented by one and the web page counter loop is reentered. If the current web page number equals the high watermark at 202, then the web page counter is reinitialized to the first web page, so that the web page loop may repeat itself, from the first internal web page, depending upon the delay setting for the last web page.

A test is then made on all objects defined for this web page at 203, utilizing a loop whose range is defined by the number of objects per web page high watermarks. More specifically, within this universe of possible objects, if the object exists, and it is defined by a time line in which there is a delayed entrance, then a boolean flag is set for those objects that causes the draw system to suppress drawing these objects during the web page transition as defined below.

A test is then made to determine if the web page has a transition animation defined at 204. If not, the draw system is called for the first time. The draw system for a given web page utilizes a loop whose range is defined by the number of objects per web page high watermarks. The draw system can also employ technology so that the draw process generates a screen image in one or more off-screen buffers, only drawing to the screen when the screen image, or the clipping area for the screen, has been fully generated. This greatly reduces, if not totally eliminates, any screen flicker, and creates visually smooth animation effects.

The first draw function is to draw the web page background into the primary off-screen buffer. The web page

US 6,546,397 B1

49

background color is drawn, as defined initially at FIG. 21a, or modified for that particular web page at FIG. 20. A test is then made to determine if the web page has a background image pattern, as defined initially at FIG. 21a, or modified for that particular web page at FIG. 20. If it does, and the image observer reports that the image is ready to be drawn, a background image draw loop is executed, defined by the height and width of the background image, and the screen resolution of the current browser window. In the unlikely event that the background image pattern is not yet available, there is a delay until the image observer reports the completion of the image processing operation. The tiled background image pattern is also drawn into the primary off-screen buffer, completely overdrawing the background color. The backgrounds for all non-suppressed (See 203) parent web page text button and paragraph objects are then drawn into the primary off-screen buffer, unless a background transparency flag has been set (See FIG. 7).

The text strings for non-suppressed parent web page text button and paragraph objects are then drawn into the primary off-screen buffer. These text strings are drawn based on their font name, style, size, effect(s), and color. If a paragraph line string, the string may have multiple string segments, each with their own font name, style, etc. If the text button object has its vertical attribute set to true, then the draw system executes a loop defined by the number of characters defined in the text button object. The top and left origin coordinates were set in the usual way (See FIG. 30), but the top coordinate is adjusted by the font height for each iteration of this draw loop. The intelligent 3D Frame, if defined, is then drawn into the primary off-screen buffer for the paragraph and text button objects (See FIG. 30). The primary image objects for the web page are then processed by the draw system. If the image observer reports that the image is ready to be drawn, it is drawn into the primary off-screen buffer, based on the coordinates and size as defined in FIG. 30. If not ready, there is a delay until the image observer reports the completion of the image processing operation. The Intelligent 3D frame, if defined, is then drawn into the primary off-screen buffer for the image objects (See FIG. 30).

The draw system is responsive to two other technologies at this stage. The first is user interaction based on the location of the mouse cursor and any user initiated mouse event. This subject will be described in greater detail below in association with FIG. 36. The second is object animation for non-delayed web page objects. This subject will be described in greater detail below in association with FIG. 33.

If the web page transition test at 204 was true, then the run time engine's main run method executes the web page transition animation technology at 205.

FIG. 32 describes the web page transition animation technology. First a lock is placed on this method at 212, as a safety precaution to prevent any interference from other threads during the animation. A test is then made on whether the transition animation setting (See FIG. 20) for the web page is random at 213. If so, a random transition number is generated at 214. The web page thread then begins a particular animation loop at 215, depending upon the random number that was generated at 214 or by the transition animation that was set previously (at FIG. 20). In one implementation, 13 different transition animations plus random are supported including. They are: Fade In, Zoom In, Zoom Out, Zoom to Upper Left, Zoom to Lower Right, Rotate to the Left, Rotate to the Right, Rotate Bottom to Top, Rotate Top to Bottom, Slide to the Left, Slide to the Right, Slide Bottom to Top, and Slide Top to Bottom.

50

For all web page transition animations, the X and Y animation increment values are calculated by dividing the current browser's screen width and height by the user defined animation resolution at 215. In all animation and draw loops, the number of loops can equal the number of animation frames as set at FIG. 20. The timer delay for all animations, in milliseconds, can be calculated by dividing the number of frames per second (See FIG. 20) into 1,000.

For a description of "Fade In" Technology see FIG. 33. A "Zoom In" algorithm sets the initial scaled width and height for the current web page image to zero and the prior web page image to its full size. In each animation and draw loop the previous web page's final image state is drawn into a secondary off-screen buffer at 216. (If this is the first occurrence of the first web page, then the secondary off-screen buffer is set to the background of the first web page.) The upper left hand corner (origin) of the current web page can be calculated based on the following formula: browser screen width minus the scaled width divided by two.

The scaled image of the current web page is then drawn into the secondary off-screen buffer at the calculated origin, using the current scaled width and height for the web page image. This merged image of the prior and scaled version of the current web page is then drawn to the screen. A timer delay then occurs as defined at 215, after which the X and Y animation increment values are added to the scaled width and height for the current web page image. The animation loop is then repeated to its conclusion at 218.

The other eleven web page transition animations follow a similar methodology, but have quite different calculations, which are based on the following variables:

- 1: Order of drawing of the prior and current web pages.
- 2: Initialization values for the X and Y origin coordinates for the current and prior web pages.
- 3: The initial values for the scaled width and height for the current and prior web pages.
- 4: Whether X and Y origin coordinates for the current and prior web pages increment, decrement, or remain the same.
- 5: Whether the values for the scaled width and height for the current and prior web pages increment, decrement, or remain the same.

For the "Zoom Out" animation, the current page is drawn first and always drawn at 100%. The prior web page is initialized also at 100%, but its X and Y origin coordinates are incremented and its scaled width and height values are decremented, by the appropriate values, for each animation iteration.

For the "Zoom to Upper Left", "Zoom to Lower Right", "Rotate to the Left", "Rotate to the Right", "Rotate Bottom to Top" and "Rotate Top to Bottom" animations, a common data initialization and data increment strategy is implemented.

- 1: The X and Y variables for page image one is set to zero.
- 2: The X and Y variables for page image two is set to the right and bottom edges of the browser window.
- 3: The scaled width and height variables for page image one is set to 100% of the browser window's resolution.
- 4: The scaled width and height variables for page image two is set to zero.
- 5: For each loop iteration, the scaled width and height variables for page image one are decremented by the X and Y animation increment values defined at 215.
- 6: For each loop iteration, the scaled width and height variables for page image two are incremented by the X and Y Animation increment values defined at 215.



US 6,546,397 B1

51

For the "Zoom to Upper Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. (upper left corner of the browser window) Its scaled width and height values are always set to the current values for scaled width and height variables for page image one. The X and Y origin coordinates for the current web page can be calculated by subtracting the current values of image two's scaled width and height variables from the initial values of the X and Y variables for page image two. The scaled width and height values for the current web page can be set to the current values for the scaled width and height variables for page image two.

For the "Zoom to Lower Right" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width and height values are always set to the current values for scaled width and height variables for page image two. The X and Y origin coordinates for the prior web page are set to current values of image two's scaled width and height variables. The scaled width and height values for the prior web page are set to the current values for the scaled width and height variables for page image one.

For the "Rotate to the Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to current value of image one's scaled width variable. Its scaled height value is always set to the bottom of the browser's window. The X origin coordinate for the current web page can be calculated by subtracting the current value for image two's scaled width variable from the initial value for image two's X origin coordinate. The Y origin coordinate for the current web page is always set to zero. Its scaled width value is set to current value of image two's scaled width variable. Its scaled height value is always set to the bottom of the browser's window.

For the "Rotate Bottom to Top" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to the width of the browser window. Its scaled height value is set to current value of image one's scaled height variable. The current web page's X origin coordinate is always set to zero. The Y origin coordinate is calculated by subtracting the current value of image two's scaled height variable from the initial value for image two's Y origin coordinate. Its scaled width value is always set to the right edge of the browser's window. Its scaled height value is set to current value of image two's scaled height variable.

For the "Rotate Top to Bottom" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to the width of the browser window. Its scaled height value is set to current value of image two's scaled height variable. The prior web page's X origin coordinate is always set to zero. The Y origin coordinate is set to the current value of image two's scaled height. Its scaled width value is always set to the right edge of the browser's window. Its scaled height value is set to current value of image one's scaled height variable.

For the "Slide to the Left", "Slide to the Right", "Slide Bottom to Top" and "Slide Top to Bottom" transition animations, a common data initialization and data increment strategy is implemented. The strategy includes

- 1: The X and Y variables for page image one is set to zero.
- 2: The X and Y variables for page image two is set to the right and bottom edges of the browser window.
- 3: For each loop iteration, the X and Y variables for page image one are incremented by the X and Y animation increment values defined at 215.

52

- 4: For each loop iteration, the X and Y variables for page image two are decremented by the X and Y animation increment values defined at 215.

- 5: The scaled width and height values always remain at 100% of the browser windows width and height.

For the "Slide to the Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. The current web page's X origin coordinate is set to the current value of page image two's X variable. Its Y origin coordinate is always set to zero. For the "Slide to the Right" Animation, the current web page is drawn first, with its X and Y origin coordinates always set to zero. The prior web page's X origin coordinate is set to the current value of page image one's X variable. Its Y origin coordinate is always set to zero.

For the "Slide Bottom to Top" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. The current web page's Y origin coordinate is set to the current value of page image two's Y variable. Its X origin coordinate is always set to zero.

For the "Slide Top to Bottom" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. The prior web page's Y origin coordinate is set to the current value of page image one's Y variable. Its X origin coordinate is always set to zero.

After the last animation cycle is completed for any of the transition animations at 218, the animation process is unlocked, and process step 206 shown in FIG. 31 is then executed.

Returning to FIG. 31, the main web page thread's run method then executes a text button and image object time line, transformation and animation loop at 206. This range loop is defined from the first object on the given web page to the high watermark for the number of those objects on a web page for this web site. A test is made on each object on whether an animation, transformation and/or time line has been assigned at 208.

If so, an "instance" of the time line class for that particular object type is created at 209. An "instance" of a class is a fundamental aspect of object oriented programming (OOP). Each time, the line class is implemented with a "runnable" interface, so that they can be executed as independent threads. Communication of data, between the "instance" of a class and the main run engine class can be accomplished in OOP using several different techniques. In one implementation, this construction, passed as an argument, is used to permit different objects to address each other's variables and databases. A thread is then created, utilizing a two-dimensional object internal database architecture (web page number by internal object number). This methodology is convenient for permitting all object time lines for a given web page to be managed and synchronized. The object's thread is then "started".

The result of this process at 209 is that an independent thread has been created for each appropriate object on a given web page, all executing simultaneously with each other and with the main run time engine web page thread, subject to the definitions of their independent time lines at 210. See FIG. 33 for a description of the time line technology. When the main web page thread has finished the text and image loop at 207, the draw system is activated; the run time engine can now respond to user interactions, and the main web page thread transitions into a "Join" loop at 211. See FIG. 35 for a description of this process.

FIG. 33 shows the time line technology used by the run time engine. The techniques and algorithms employed to create this technology permit each web page object to have

US 6,546,397 B1

53

an independent yet synchronized existence with each other, with the main web page thread, and with child objects that each main or parent object may spawn. Furthermore, each object and each of their child objects are capable of performing multiple animations and transformations, either serially or simultaneously. Database initialization is first accomplished for each object thread. This assures that the object thread's database is set to the correct initial values as required for that particular object, and that the references to the main web page thread's database are established.

A test is then made to determine if the object has a time line definition assigned to it at 219. If not, a test is made at 220 on certain two-dimensional object definition variables in order to determine which of the following four states have been defined for the object: animation without a transformation; transformation without animation; animation, with the transformation occurring simultaneously with the animation; and animation and transformations occurring in a serial manner.

If the test shows that the object has an animation defined, but no transformation, then certain two-dimensional status variables are set, and an "instance" of the "animation class" for that particular object type is created at 229. Each "animation class" is also implemented with a "runnable" interface. An object animation thread is then created, utilizing the two-dimensional object internal database architecture (See FIG. 8). This object animation thread is then "started". Communication between the object animation thread, the parent time line thread, and its parent, and the main web page thread, are accomplished as discussed in process 209. The object time line thread then executes a "Join" method. This puts the object time line thread in a "wait state". When the thread it is waiting for is completed, this child thread "joins" the parent object time line thread, and the object time line thread then continues its process. Other forms of synchronization between two independent threads could have been implemented as is known in the art.

The techniques employed at 229 to implement object animation vary by object type. In one implementation, for text button object animations, 26 different animations are supported including: Zoom In, Zoom Out, Grow NW, Grow NE, Grow SE, Grow SW, Shrink SE, Shrink SW, Shrink NW, Shrink NE, Enter N, Enter NE, Enter E, Enter SE, Enter S, Enter SW, Enter W, Enter NW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW. In one implementation, for image object animations, 29 different animations are supported including: Fade In, Fade out, Rotate, Zoom In, Zoom Out, Grow NW, Grow NE, Grow SE, Grow SW, Shrink SE, Shrink SW, Shrink NW, Shrink NE, Enter N, Enter NE, Enter E, Enter SE, Enter S, Enter SW, Enter W, Enter NW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW.

As discussed above with regard to FIG. 17 each animation type has a defined speed, resolution, and number of animation cycles. These settings are stored in the main web page class, and are passed to the particular animation thread through a two-dimensional object internal database architecture as discussed in process step 209 above during the animation thread's initialization process. The animation thread then executes, in its run method, a main animation loop that has the number of iterations set to the end number of animation cycles, as assigned to that particular text button object.

Text button animations are currently implemented in three logical groups. Group One includes "Zoom In", "Grow NW", "Grow NE", "Grow SE", and "Grow SW". Group Two includes "Zoom Out", "Shrink SE", "Shrink SW",

54

"Shrink NW", and "Shrink NE". Group Three includes "Enter N", "Enter NE", "Enter E", "Enter SE", "Enter SW", "Enter W", "Enter NW", "Exit N", "Exit NE", "Exit E", "Exit SE", "Exit S", "Exit SW", "Exit W" and "Exit NW".

For Group One text button animations, the animation font size is initialized at a very small value, and in one implementation is set at 4 Points. The animation point size increment can be derived by dividing the resolution (number of animation frames) into the font size for that text button object. The run method then executes a secondary animation loop, which will terminate when the animation point size equals the text button object point size. For each secondary animation loop, the length of the current animated text string is calculated, a new font object is created for the current animation point size, and the font metrics for that new font are created. If the text button object has a vertical orientation, the animated text button object's width is calculated to be the width of an "m" space, in the current animated font. The animated text button object's height is calculated to be current animated font height times the number of characters in the text string. If the text had a horizontal orientation, the animated text button object's width is calculated to be the width of the text string in the current animated font. The animated text button object's height can be calculated to be the font height of the current animated font. The calculations for X and Y coordinates for the animated text button object depend upon which animation was defined within the Group One-text button animations. The X and Y animation increments can be calculated utilizing the height and width, in pixels, of the text button object scaled to the current browser's window, utilizing the text button animation resolution, and considering whether the animating text button object is being centered during the animation ("Zoom Out") or not. These calculations are similar to those for the web page transition animations discussed with regard to FIG. 32.

The draw system is then called. Based on the values of the two-dimensional status variables that had been set initially, the draw system executes the appropriate animation draw routine utilizing, through the data communication techniques already discussed, the current animation font point size, and the current animation X and Y coordinates. If a text background is to be drawn, the same algorithm as defined in FIG. 31 is used. If a 3D Frame is assigned, the current animated string width and height are passed to the appropriate 3D frame generation method, and the frame is drawn with the same algorithm as defined in FIG. 31, but utilizing the current animation X and Y coordinates. The text button object's orientation is also handled by the draw system with the same algorithms as defined in FIG. 31.

The text button animation thread then executes a timer delay, whose value had been defined in FIG. 17. When the timer reactivates the text button animation thread after the appropriate delay, an animation cycle completion test is made to see if the text button object's point size minus the animation point size is less than the animation point size increment. This type of testing methodology permits the invention to utilize integer values, as opposed to floating point values, for the text button animation. This improves the execution of the animation considerably.

If the above test is true, the animation point size is set equal to the object point size and a final call is made to the draw system for that animation cycle. A test is then made to see if the current animation cycle equals the total number of animation cycles as defined in FIG. 17. If not, a new animation cycle is initiated, with the animation values reinitialized. If this was the last animation cycle the text

US 6,546,397 B1

55

button animation thread calls its "stop" method, which sets the required status variables as appropriate, then terminates itself. This causes the parent text button time line thread to be reactivated through the "join" mechanism.

If the results of animation cycle completion test are false, the current animation point size is increased by the animation point size increment. A new font object is created for the now current animation point size, and new font metrics for that new font are created. If the text button object has a vertical orientation, the animated text button object's width is calculated to be the width of an "m" space, in the now current animated font. The animated text button object's height is calculated to be the now current animated font height times the number of characters in the text string. If the text has a horizontal orientation, the animated text button object's width is calculated to be the width of the text string in the now current animated font. The animated text button object's height is calculated to be the font height of the now current animated font. The calculations for the new X and Y coordinates for the animated text button object are then completed, as appropriate, and the draw system is called again.

The algorithms for Group Two text button animations are very similar to those of Group One. The differences are just in what are the initial animation values, and whether the animation point size increments and the animation X and Y coordinate increments are added or subtracted from the then current animation point size and the then current X and Y coordinates for the animating text button object.

For Group Three text button animations, the distance that the text button animation will move is calculated, in pixels, from its initial location to its final location in the current browser window. The X and Y animation increments are calculated by dividing that distance by the resolution of the text button animation. All the other algorithms for Group Three text button animations are generally a subset of those for Group One, and similar to the web page slide transition animations defined with reference to FIG. 31.

Referring again to FIG. 33, image animations at process step 229 can currently be grouped into five logical classes. As with text button animations, Group One includes "Zoom In", "Grow NW", "Grow NE", "Grow SE", and "Grow SW". Group Two includes "Zoom Out", "Shrink SE", "Shrink SW", "Shrink NW", and "Shrink NE". Group Three includes "Enter N", "Enter NE", "Enter E", "Enter SE", "Enter S", "Enter SW", "Enter W", "Enter NW", "Exit N", "Exit NE", "Exit E", "Exit SE", "Exit S", "Exit SW", "Exit W" and "Exit NW". In addition, image animations have a Group Four, which includes "Fade In" and "Fade Out". Group 5 image animations include the "Rotate" Animation.

For Group One image animations, the animation width and height increments are calculated by dividing the image object's width and height by the resolution (number of animation frames) as set in FIG. 17. The initial animation width and animation height values are set to a very small number, currently equal to the animation width and height increment values just calculated. The calculations for X and Y coordinates for the animated image object depends upon which animation was defined within the Group One text button animations. The X and Y animation increments are calculated utilizing the height and width, in pixels, of the image object scaled to the current browser's window, utilizing the image animation resolution, and considering whether the animating image object is being centered during the animation ("Zoom Out") or not. These calculations are similar to those for the web page Transition Animations discussed above with regard to FIG. 32.

56

The run method then executes a secondary animation loop, which will terminate when the animation width equals the image object's width. The algorithms employed by the invention to change the animating object's height, width, X coordinate, and Y coordinate are very similar to those employed for Group One text button animations, and will not be repeated here. The techniques to utilize the draw system for drawing the image animation, the time delay technique, and the post draw logic tests and actions are also very similar.

The algorithms for Group Two image animations are very similar to those of Group One. The differences are just in what are the initial animation values, and whether the animation width and height increments and the animation X and Y coordinate increments are added or subtracted from the then current animation width and height and the then current X and Y coordinates for the animating image object.

For Group Three image animations, the algorithms are identical to those of Group Three text button animations. For Group Four image animations, the "alpha" value of a given image object is utilized in order to implement "Fade In" and "fade Out" animations. The alpha value can range from 0 to 255, depending upon the image strength desired. The value for an alpha animation increment variable can be calculated by dividing the resolution of the animation into 255, after making the necessary adjustments to keep the data in integer form, without losing resolution due to integer rounding errors. For a "Fade In" animation the value of an alpha animation variable is set to zero. The run method then executes a secondary animation loop, which will not terminate until 255 minus the then current value of the alpha animation variable is less than the value alpha animation increment variable. A "Fade In" image filter can be created for each iteration of the animation loop, using the current setting of the alpha animation variable. An image producer can also be created with pointers to the last image bitmap produced for the image object in the last animation loop and to the image filter that has just been created. The image producer, under the control of a media tracker then creates a new image bitmap. The animation thread then "waits" for the completion of this image-processing event using the media tracker. Upon completion, the draw system is called which draws the then current state of the image object. The image animation thread goes into a timer delay of some preset value (in one implementation 500 milliseconds), to permit a smooth visual animation effect. The value for the alpha animation increment is added to alpha animation variable and the loop is then repeated until the loop condition is met. Then the "stop" method is called, certain status variables are set, and the image animation thread terminates itself. This causes the parent image time line thread to be reactivated through the "Join" mechanism.

The "Fade Out" animation employs very similar technology, except that:

- 1: the alpha animation variable is set to zero,
- 2: the value for the alpha animation increment is subtracted, and
- 3: the loop termination test is when the value for the alpha animation variable is less than the value for the alpha animation increment.

For the Group Five image rotate animation, a different bitmap for the image object is created for each animation frame through the use of a progression of standard geometrical transformations on the original image bitmap. A secondary animation loop is then executed as defined by the number of animation frames. In each loop iteration, an image object is created from an appropriate image bitmap



US 6,546,397 B1

57

selected from among the set just created, the necessary two-dimensional variables are set to communicate with the draw system, and the draw system is then called. The image animation thread then executes a timer delay method based on the delay setting as defined above with reference to FIG 17. When the timer reactivates the image animation thread after the appropriate delay, the next iteration of the secondary animation loop is repeated until the loop condition is met. Then the "stop" method is called, certain status variables are set, and the image animation thread terminates itself. This causes the parent image time line thread to be reactivated through the "join" mechanism.

Returning to process step 220 shown in FIG. 33, if the object had a transformation, but not an animation, then certain two-dimensional status variables are set, and an "instance" of the "transformation class" for that particular object type is created at 228. Each "transformation class" is also implemented with a "runnable" interface. An object transformation thread is then created, utilizing the invention's two-dimensional object internal database architecture. This object transformation thread is then "started". The inter-thread communication technology and the "join" technology employed for object transformations is the same as for object animations.

If the transformation is being applied to a text button object at 228, then a timer delay method is executed based on the delay setting as described in association with FIG. 18. When the timer reactivates the text button transformation thread after the appropriate delay, the appropriate two-dimensional status variables are set to inform the draw system which state of the current text button object to draw. The draw system is called and, based on the settings for the above mentioned two-dimensional status variables, either the "normal", mouse over", mouse down" or "pop-up" states of the text button object's background, if any, the text button object's string, and the 3D frame, if any, are drawn. If additional transformations are defined (FIG. 18), the above process is repeated, based on the timer delay and object states defined for the subsequent transformations. When the last transformation is completed, the "stop" method is called, which sets the required status variables as appropriate. This causes the parent text button time line thread to be reactivated through the "join" mechanism.

If the transformation is being applied to an image object at 228, then a timer delay method is executed based on the delay setting (as defined in FIG. 18). When the timer reactivates the image transformation thread after the appropriate delay, image transformation technology is executed. In one implementation, the image transformation technology utilizes the "alpha" value of a given image object state in order to fade in and fade out images. The alpha value can range from 0 to 255, depending upon the image strength desired. The value for an alpha transformation increment variable is calculated by dividing the resolution of the transformation into 255, after making the necessary adjustments to keep the data in integer form, without losing resolution due to integer rounding errors. The value of an alpha transformation variable is set to zero. Depending upon the settings as defined in FIG. 18, the bitmap for one image object state is initialized to an alpha value of zero, while another is initialized to an alpha value of 255. The appropriate two-dimensional status variables are set for communication with the draw system.

A transformation loop is then executed, until 255 minus the then current value of the alpha transformation variable is less than the value alpha transformation increment variable. This methodology again keeps all calculations in the form of

58

integers, as opposed to floating point, thus speeding up the transformation process.

Two "Fade In" image filters are created for each iteration of the transformation loop. The first uses an alpha value calculated at the current setting of the alpha transformation variable. The second uses an alpha value calculated at 255 minus the current setting of the alpha transformation variable. Two image producers are also created with pointers to the last image bitmap produced for each image object state in the last transformation loop and to the two image filters that had just been created. The two image producers under the control of two media trackers then create two new image bitmaps. The transformation thread then "waits" for the completion of these two image processing events using the media trackers. Upon completion, the draw system is called which draws the then current state of the two image object states, in the correct order, and in the correct location. The image transformation thread goes into a timer delay of some preset value (500 milliseconds in one implementation), to permit a smooth visual transformation effect. The loop is then repeated until the loop condition is met. Then the "stop" method is called, certain status variables are set, and the image transformation thread terminates itself. This causes the parent image time line thread to be reactivated through the "join" mechanism.

Returning to process step 220 in FIG. 33, if the object was defined with an animation and transformation that would execute in a serial manner, then certain two-dimensional status variables are set, and an "instance" of the "transformation" class for that particular object type is created at 230. An object transformation thread is then created, utilizing the two-dimensional object internal database architecture. This object transformation thread is then "started" and the parent object time line thread "waits" to be "joined".

If a text button object, then a primary loop is executed, with the number of iterations set to the number of transformations. After the execution and return from a timer delay event, if any, an "instance" of the text button animation class is created, and then a text button animation thread is created and "started". The parent text button transformation thread then waits to be "joined". This causes the text button animation thread to be executed, in the manner described at 229. When the text button animation thread completes its execution, it calls its "stop" method, which sets the necessary status variables and then terminates itself. This causes the text button animation thread to "join" the parent text button transformation thread, causing that thread to resume processing. The first text button transformation is then executed, in the manner described at 228. After the execution and return from another timer delay event, if any, another "instance" of the text button animation class is created, and then another text button animation thread is created and "started". The parent text button transformation thread again waits to be "joined". This causes the text button animation thread to be executed again with the animation being executed, based on the definition set at FIG. 18, on a different text button object state. The loop is then repeated until the last text button transformation is completed. Then the text button transformation thread calls its "stop" method, certain status variables are set, and the text button transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the "join" mechanism.

If an image object, the mechanism of the image transformation thread spawns image animation threads, before each transformation, and is the same as that of a text button object. The actual image transformation process is identical

US 6,546,397 B1

59

to that described at 228. When completed the "stop" method is called, certain status variables are set, and the image transformation thread terminates itself. This causes the parent image time line thread to be reactivated through the "join" mechanism.

Returning to process step 220 in FIG. 33, if the object was defined with a simultaneous animation and transformation, then certain two-dimensional status variables are set, and an "instance" of the "super transformation class" for that particular object type is created at 231. In one implementation, the animation, transformation, and super transformation classes are integrated into one structure in order to reduce code size and increase execution speed. Each "super transformation class" is also implemented with a "runnable" interface. An object super transformation thread is then created, utilizing the two-dimensional object internal database architecture. This object super transformation thread is then "started". The inter-thread communication technology and the "join" technology employed for object super transformations is the same as for object transformations.

If a text button object, a calculation is made in order to prorate the text button animation process across the defined text button transformation process. The calculation is driven by the number of text button animation frames, and prorates from that total the number of frames that should be assigned to each transformation state. This can be done by dividing the sum of all the transformation times by each individual transformation time, and multiplying that result by the number of frames, making necessary adjustments to prevent integer rounding error. After these calculations are completed, the text button animation is executed in a similar manner as was defined at 229. However, when the appropriate number of animation frames had been drawn, certain two-dimensional status variables are set prior to calling the draw system for the next animation frame, so that the correct text button object state is drawn, in the correct size and with the correct coordinates, by the draw system. When the super transformation process is completed the "stop" method is called, certain status variables are set, and the text button super transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the "join" mechanism.

If an image object, a calculation is made in order to prorate the image animation process across the defined image transformation process. The calculation is driven by the number of image transformation events that would occur (where each one can be set at approximately 500 milliseconds) over the entire animation event. A calculation is performed in order to calculate how many image transformation events should be assigned to each transformation state. This is done by dividing the sum of all the transformation times by each individual transformation time, and multiplying that result by the total number of transformation events, making necessary adjustments to prevent integer rounding error. A calculation is then made to allocate the number of animation frames to each image transformation event. After these calculations are completed, the image animation is executed in a similar manner as was defined at 229. However, when the appropriate number of animation frames had been drawn, the image transformation technology is called to perform the next transformation event. The alpha transformation increment can be defined by dividing 255 by the number of transformation events assigned to that transformation. The draw system is then called. When the number of image transformation events assigned to a given image transformation is reached, then certain two-dimensional status variables are set prior to calling the draw

60

system for the next animation frame, so that the correct image states, in the correct size and with the correct coordinates, are utilized by the draw system. This entire animation/transformation process will be repeated by the number of image animation cycles. When the super transformation process is completed the "stop" method is called, certain status variables are set, and the text button super transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the "join" mechanism.

Returning to process step 219 in FIG. 33, if the object had a time line, then a test is made at 221 on whether an appearance delay had been defined in FIG. 19. If so, a timer event is set at 222. When the timer reactivates the object time line thread after the appropriate delay, a test is made on whether an entry animation/transformation has been defined for this object time line at 224, as described FIG. 19. If so, based which animation/transformation process was defined, it is created and executed at 228, 229, 230, or 231. In one implementation, 13 entry animations are supported for both text button and image objects, and an additional "Fade In" entry animation is supported for image objects. The 13 common entry animations supported include Zoom In, Grow NW, Grow NE, Grow SE, Grow SW, Enter N, Enter NE, Enter E, Enter SE, Enter S, Enter SW, Enter W and Enter NW.

If no entry animation/transformation is defined, or when the entry animation/transformation has "joined" the object time line thread, a test is made to determine if any child time lines have been defined at 225, as described in FIG. 19, for this parent object time line. If so, an "instance" of the "child time line class" for that particular object type is created at 226. Each "child time line class" is also implemented with a "runnable" interface. An object child time line thread is then created, utilizing the two-dimensional object internal database architecture. This object child time line thread is then "started". The inter-thread communication technology and the "join" technology employed for object child time lines is the same as for object time lines. Either a text button child time line thread or an image child time line thread, or both, can be spawned at this time. Simultaneous with the execution of any spawned text button child time line threads, the parent object thread then executes the defined main animation and or transformation. As with non-time line object threads, a test is made on certain two-dimensional object definition variables in order to determine which of the following four states have been defined for the object at 227: animation without a transformation; transformation without animation; animation, with the transformation occurring simultaneously with the animation; and, animation and transformations occurring in a serial manner.

Based on the results of this test, an appropriate "instance" of an appropriate animation, transformation, or super transformation class is created, and an appropriate animation, transformation, or super transformation thread is created and "started". This results in the execution of process steps 228, 229, 230, or 231, as defined above.

The parent object time line thread then executes a "join" method. This again puts the object time line thread in a "wait state". When the thread it is waiting for is completed, the child thread "joins" the parent object time line thread, and the object time line thread then continues its process. The object time line thread then checks to see if there is a departure delay defined at 232. If so, it sets a timer event at 233. When the timer reactivates the object time line thread after the appropriate delay, a test is made at 234 on whether an exit animation/transformation has been defined for this



US 6,546,397 B1

61

object time line, as described in FIG. 19. If so, it is created at 235, and performed as discussed with reference to processes 228, 229, 230, or 231. In one implementation, 13 exit animations supported for both text button and image objects, and an additional "Fade Out" exit animation is supported for image objects. The 13 common exit animations include: Zoom Out, Shrink NW, Shrink NE, Shrink SE, Shrink SW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW.

If no exit animation/transformation is defined, or when the exit animation/transformation has "joined" the object time line thread, the parent object time line thread then executes a "join" method if it had spawned any child time lines. This again puts the object time line thread in a "wait state". Finally, when then the child time line threads, if any, "join" the parent object time line, the "stop" method for the parent time line is called. Certain status variables are set, and the parent object time line thread terminates itself. This causes the main web page time line that had been in a "join" loop at 211 of FIG. 31, since the invocation of the object time lines, to be "joined" by this particular object time line thread.

FIG. 34 shows the technology employed by the run time engine for implementing child time lines for text button and image objects. Child text button object time lines and child image object time lines are subsets of their parent object time lines. First a test is made at 237 on whether an appearance delay had been defined (See FIG. 19). If so, a timer event is set at 238. When the timer reactivates the child object time line thread after the appropriate delay, a test is made on whether an entry animation has been defined for this child object time line at 239 (as described FIG. 19). If so, it is created and executed at 240 in a manner identical to that described at process step 229 in FIG. 33. The same 13 entry animations supported for parent object time lines are also supported for both child text button and image objects, and the additional "Fade In" entry animation is supported for child image objects. The "join" mechanism described in FIG. 33 is employed in an identical manner at 240 to synchronize the child time line thread with its entry animation thread.

After being "joined" and reactivated, the child object time line performs a test at 241 on whether an exit delay had been defined (See FIG. 19). If so, a timer event is set at 242. When the timer reactivates the child object time line thread after the appropriate delay, a test is made on whether an exit animation has been defined for this child object time line at 243, as described in association with FIG. 19. If so, it is created and executed at 244 in a manner identical to that described above at process step 229 in FIG. 33. The same 13 exit animations supported for parent object time lines are also supported for both child text button and image objects, and the additional "Fade Out" exit animation is supported for child image objects. The "join" mechanism described above in association with FIG. 33 is employed in an identical manner at 245 to synchronize the child time line thread with its parent object time line thread. As discussed at process step 236 in FIG. 35, the parent object time lines "wait" until all their child time lines have terminated, before they in turn terminate and "join" the main web page time line at FIG. 35.

FIG. 35 describes technology employed by the run time engine for the web page and object thread loop. As noted in FIG. 31 at process step 211, after all the text button and image time line threads for the current web page had been launched, the main web page thread executed a "join" loop, waiting for the completion of all the parent object time line

62

threads. Because each parent object time line thread waited for their child object time line threads to be "joined", as well as any other spawned animation threads, transformation threads, and/or super transformation threads, the effect of this "join" loop at 246 is that the web page thread will not resume processing until all parent time line threads have completed and that of all of their spawned threads.

Upon resuming its processing after the "join" process at 246 has been completed, the main web page thread checks at 247 to see whether the current web page has an automatic termination, based on a timer delay, or whether the web page will wait for a user interaction before terminating. If the web page has a time delay based termination setting, then a timer method is called at 249, and the web page goes to "sleep" awaiting the completion of the timer event.

When the timer event occurs, the web page thread resumes processing by incrementing the web page counter by one, and the entire web page process, which began at process step 200 in FIG. 31, is repeated. If the current web page termination setting was to set to wait until user interaction, then web page thread is placed in a "pause" state, and the run time engine waits to respond to any mouse, keyboard or other user initiated event.

FIG. 36 describes the technology employed by the run time engine for responding to user interactions. As mentioned in association with process step 204 of FIG. 31, as soon as the draw system has been activated, the run time engine will respond to any user interactions that have been defined (See FIG. 16). This is also true during any object time line events, as with respect to process step 207 of FIG. 31. The run time engine currently responds to "mouse over" and "mouse down" events for text button, image, and paragraph objects. For form objects, the run time engine will also respond to keyboard events. As the full-featured programming languages supported by browsers evolve, the run time engine may be configured to respond to other user interactions, including but not limited to single and double clicks from both the left and right mouse button, voice commands, eye focusing technologies, touch screen technologies, and push technologies originating from a server.

The run time engine invokes a "dynamic mouse to object recognition" technology at 251 in order to be responsive to the following elements:

- 1: The location of objects will vary based on the viewer's screen resolution and browser window size as discussed above with regard to FIG. 27.
- 2: Objects may move or resize themselves based on time lines and animations.
- 3: Objects may have different sizes based on the state they are being displayed in based on time lines and transformations.
- 4: More than one object can occupy the same screen location, and which objects occupy that location may change over time based on time lines, animations, and transformations.

The run time engine maintains, in its internal database, the object's current X and Y origin coordinates, and the object's current width and height, in pixels, based on the current viewer's screen and browser window size. This can be accomplished by first converting all coordinates and sizes to the current viewer environment with the scaling technology as discussed above with regard to FIG. 27. Every time line, animation, and transformation thread updates, in real time, the run time engine's internal database positional and size variables of the objects they define, utilizing the data communication techniques described above with reference to FIG. 33.

US 6,546,397 B1

63

The run time engine employs mouseEnter, mouseMove, mouseDown, mouseDrag, mouseUp, and mouseExit methods to constantly monitor the state of the mouse at all times. The onClick method (to detect a single click) and a specialized method to detect a double click event are also employed. The onKeyDown method, with processing the returned scan code, permits the run time engine to process all keyboard events. The mouseEnter, mouseMove, mouseDown, mouseDrag, mouseUp, and mouseExit methods return to the run time engine the exact X and Y coordinates of the mouse cursor at the instant that particular mouse event occurred. Thus for each supported mouse based user interaction technique supported by the run time engine, a two-dimensional loop exists (web page number by object number) in which the current bounding rectangle for every object on a given web page is being compared to the current mouse cursor location at all times. The bounding rectangle is simply the current X and Y origin coordinates of an object, extended by its current width and height. In this way, the run time engine is informed if the current mouse cursor location falls within one or more bounding rectangles. Parenthetically, the run time engine also knows when the current mouse cursor falls outside the bounding rectangle of a given object.

Based on the type of mouse user interaction at 252, the run time engine employs different techniques and executes different methods. If the viewer moves the mouse at 253, the mouseMove method informs the run time engine immediately of this event and the current mouse cursor coordinates.

If this user mouse move action caused the mouse to move into one or more bounding rectangles of any text button or image object(s) at 254, or out of one or more bounding rectangles of any text button or image object(s) at 255, then appropriate two-dimensional status variables are set and the draw system is called. The draw system interprets the relevant two-dimensional variables for the existing text button and image object(s) on the current web page in its draw loop as described above with reference to FIG. 16, and draws the correct backgrounds, text strings, images, and 3D frames based on the state of each object, as just set by the "mouseMove" computational two-dimensional loop. If the mouse has entered into or out of the bounding rectangles of any image and/or text button object that has a defined text button, image and/or video pop-up object (See FIG. 16), then the draw system paints or effectively erases the appropriate background, text string, images and/or 3D frame for these pop-up objects. If the location of any of these text button and image objects or child pop-up objects is changing over time because of time line, animation, or transformation threads, the draw system, as described in association with FIG. 33 and FIG. 34, is aware of these dynamics, and redraws the screen as these real time events occur. If any sound or video events were defined (See FIG. 16) for any text button or image objects, then the run time engine plays those sound and/or video files or channels as defined. As multiple objects can be defined that each have associated sound (and even video) files, and these objects can be overlaid on each other, either completely or partially, very interesting synchronized multiple sound tracks can be constructed, in which certain designated sounds are played, or stopped, based purely on user mouse movement.

If the viewer moves the mouse into or out of the bounding rectangle for a paragraph hot link at 256, then appropriate four-dimensional status variables (web page by paragraph number, by paragraph line number by paragraph line segment number) are set or reset and the draw system is called. The draw system paints the background color behind the hot

64

link text string, and the color for the hot link text string in the hot link "mouse over" or the hot link normal colors. The text string may be underlined or in a bold font, depending on the settings.

If the viewer presses a mouse button at 257, the mouse-down method informs the run time engine immediately of this event and the current mouse cursor coordinates. If the viewer releases the mouse at 259, the mouse-up method informs the run time engine immediately of this event.

Either way, if the original "mouse down" event had occurred inside one or more bounding rectangles of any text button or image object(s) at 258, then appropriate two-dimensional status variables are set and the draw system is called. The "mouse up" event will cause the run time engine to reset those appropriate two-dimensional status variables, and then call the draw system. If the mouse was inside of the bounding rectangles of any image and/or text button object that had a defined text button, image and/or video pop-up object (See FIG. 16) with the freeze attribute, the appropriate two-dimensional status variables are set so that the draw system will not erase these pop-up objects when the mouse moves outside the bounding rectangle(s) of the parent text button or image objects. The draw system interprets the relevant two-dimensional variables for the existing text button and image object(s) on the current web page in its draw loop described above with reference to FIG. 16, and draws the correct backgrounds, text strings, images, and 3D frames based on the state of each object, as just set by the "mouseDown" computational two-dimensional loop.

If the 3D frame had been previously defined (See FIG. 16) to have the "live" setting, then the 3D effect is changed from a "raised" effect to a "depressed" effect. If the location of any of these text button and image objects or child pop-up objects is changing over time because of time line, animation, or transformation threads, the draw system, as described in association with FIG. 33 and FIG. 34, is aware of these dynamics, and redraws the screen as these real time events occur, but does not recognize any new "mouse down" or "mouse up" Events. In one implementation, only "mouse over" events are recognized dynamically by the draw system.

If any sound or video events were defined (See FIG. 16) for any text button or image objects, then the run time engine plays those sound and/or video files or channels as defined. As multiple objects can be defined that each have associated sound (and even video) files, and these objects can be overlaid on each other, either completely or partially, very interesting synchronized multiple sound tracks can be constructed, in which certain designated sounds are played, or stopped, based purely on the user pressing a mouse button.

If a mouse down event as described previously in association with FIG. 16 was defined for one or more effected text button or image objects, only the object that was drawn last will have its event processed. If the event was to go to a different internal web page, then the run time engine sets the web page counter described in association with FIG. 31 for the "current" page to be one less than that of the desired web page. The current web page's object time lines, if any, complete normally, and the desired web page is then executed. If the "mouse down" event was to go to an external web page in a different window, then the run time engine creates a new browser window. In JAVA this can be accomplished with the "getAppletContext().showDocument(theURL, "\_blank")" method, where "theURL" is the URL address for the external web page. The run time engine, however, continues executing.

US 6,546,397 B1

65

If the mouse down event was to go to an external web page in the same browser window, then the run time engine terminates by turning control over to the designated external web page. In JAVA this can be accomplished with the "getAppletContext( ).showDocument(theURL)" method, where "theURL" is the URL address for the external web page.

If the viewer presses a mouse button while inside the bounding rectangle for a paragraph hot link at 260, then appropriate four-dimensional status variables (web page by paragraph number, by paragraph line number, by paragraph line segment number) are set. If the hot link setting at FIG. 16 was to go to a different internal web page, then the run time engine sets the web page counter described previously in association with FIG. 31 for the "current" page to be one less than that of the desired web page. The current web page's object time lines, if any, complete normally, and the desired web page is then executed. If the hot link setting was to go to an external web page in a different window, then the run time engine creates a new browser window and loads the external web page. The run time engine, however, continues executing. If the hot link setting was to go to an external web page in the same browser window, then the run time engine terminates by turning control over to the designated external web page. This completes the detailed description of the run time process at 261.

As is obvious from the descriptions of the various features and processes described above, it will be apparent to one skilled in the art that variations in form and detail may be made in the preferred implementation and methods without varying from the spirit and scope of the invention as defined in the claims or as interpreted under the doctrine of equivalents. It should also be clear that terms such as "browser", "mouse", "server", "web", etc., while adequate to describe the current state of interactive systems such as the World Wide Web, may evolve into new and more powerful entities. This evolution of terminology and technology is independent of the preferred implementation and methods of the invention as defined in the claims or as interpreted under the doctrine of equivalents. The preferred implementation and methods are thus provided for purposes of explanation and illustration, but not limitation.

I claim:

1. A method to allow users to produce Internet websites on and for computers having a browser and a virtual machine capable of generating displays, said method comprising:

- (a) presenting a viewable menu having a user selectable panel of settings describing elements on a website, said panel of settings being presented through a browser on a computer adapted to accept one or more of said selectable settings in said panel as inputs therefrom, and where at least one of said user selectable settings in said panel corresponds to commands to said virtual machine;
- (b) generating a display in accordance with one or more user selected settings substantially contemporaneously with the selection thereof;
- (c) storing information representative of said one or more user selected settings in a database;
- (d) generating a website at least in part by retrieving said information representative of said one or more user selected settings stored in said database; and
- (e) building one or more web pages to generate said website from at least a portion of said database and at least one run time file, where said at least one run time file utilizes information stored in said database to

66

generate virtual machine commands for the display of at least a portion of said one or more web pages.

2. An apparatus for producing Internet websites on and for computers having a browser and a virtual machine capable of generating displays, said apparatus comprising:

- (a) an interface to present a viewable menu of a user selectable panel of settings to describe elements on a website, said panel of settings being presented through a browser on a computer adapted to accept one or more of said selectable settings in said panel as inputs therefrom, and where at least one of said user selectable settings in said panel corresponds to commands to said virtual machine;
- (b) a browser to generate a display in accordance with one or more user selected settings substantially contemporaneously with the selection thereof;
- (c) a database for storing information representative of said one or more user selected settings; and
- (d) a build tool having at least one run time file for generating one or more web pages, said run time file operating to utilize information stored in said database to generate commands to said virtual machine for generating the display of at least a portion of said one or more web pages.

3. The apparatus of claim 2, wherein said database is a multi-dimensional array structured database.

4. The apparatus of claim 3, wherein said representative information is Boolean data, numeric data, string data or multi-dimensional arrays of various multimedia objects.

5. The apparatus of claim 4, wherein said elements include multimedia objects selected from the group consisting of a color, a font, an image, an audio clip, a video clip, a text area and a URL.

6. The apparatus of claim 2, wherein said elements are selected from the group consisting of a button, an image, a paragraph, a frame, a table, a form and a vector object.

7. The apparatus of claim 2, wherein said one or more web pages of a website is two or more web pages of a website, wherein said elements include web pages, and wherein said description of elements is a transition or an animation between two of said two or more web pages.

8. The apparatus of claim 2, wherein said elements include one or more objects on a web page, and wherein said description of elements are a transition or an animation of at least one of said elements on a web page.

9. The apparatus of claim 2, wherein said elements include a button or an images, wherein said selectable settings includes the selection of an element style, and wherein said build engine includes means for storing information representative of selected style in said database.

10. The apparatus of claim 9, wherein said elements are described by multiple object states.

11. The apparatus of claim 9, wherein said elements are described by a transformation or a timelines of said selected styles.

12. The apparatus of claim 9, wherein at least one of said elements is a child element, and wherein said element style is a child element style.

13. The apparatus of claim 12, wherein at least one of said element is described by timelines of said child elements.

14. The apparatus of claim 2, wherein said elements include buttons or images, wherein said description of elements is a transition or a timeline which is selected according to input from a mouse, and wherein said build engine includes means for storing information representative of said selected description of elements in said database.

15. The apparatus of claim 14, wherein at least one of said description of elements is a timeline or an animation.



US 6,546,397 B1

67

16. The apparatus of claim 2, wherein said elements include one or more objects on two or more web page, wherein at least one of said description of elements is a transition or a timeline, and wherein said build engine includes means to synchronize said description of said one or more objects on one web page between said two or more web pages.

17. The apparatus of claim 2, wherein one or more of said elements is a button or an image, wherein said description of elements is a transition, an animation or a timeline, and wherein said build engine includes means to synchronize said description of said one or more elements.

18. The apparatus of claim 2, wherein at least one of said elements is an object elements selected from the group consisting of a button and an image, and web pages as elements of a website, wherein said description of said object elements is a transition, an animation or a timeline, wherein said description of said web pages is a transition or a timeline, and wherein said build engine includes means to synchronize said description of said object elements and said description of said web pages.

19. The apparatus of claim 2, wherein said elements include an object and a child object, wherein said description of said elements is a timeline or an animation, and wherein said build engine activates said description of said elements according to input from a mouse.

20. The apparatus of claim 2, wherein at least one of said elements is a child button or a child object, wherein said description of said elements is a timeline, a transition or an animation, and wherein said build engine includes means for defining said description of said element.

21. The apparatus of claim 17, wherein said elements further includes at least two child object elements, and said means to synchronize includes means for synchronizing said description of at least two of said at least two child object elements.

22. The apparatus of claim 18, wherein said elements further includes a child object element, and said means to synchronize includes means for synchronizing said description of said child elements.

23. The apparatus of claim 19, wherein said description of elements is a transition or a timeline which is selected according to input from a mouse, and wherein said build engine includes means for storing information representative of said selected description in said database.

24. The apparatus of claim 2, wherein said run time files include one compressed website specific, customized run time engine program file and one compressed website specific, customized run time engine library file.

25. The apparatus of claim 24, wherein said run time files include a dynamic web page scaling mechanism, whereby each of said one or more generated web pages is scaled for viewing on said display.

26. The apparatus of claim 2, wherein said run time files includes one compressed website specific, customized run time engine program file and one compressed website specific, customized run time engine library file, wherein said run time files include a dynamic web page scaling mechanism, and wherein the build engine generates an HTML shell file for each set of one or more run time files which instructs said browser to display a background in said browser window identical to that which the run time engine will draw, invokes said dynamic web page scaling mechanism, and invokes said run time engine.

27. The apparatus of claim 2, wherein said run time files include a multi-level program animation model that presents multiple user interactions and time sensitive operations simultaneously.

68

28. The apparatus of claim 2, wherein at least one of said elements includes individual ones of multiple audio tracks, and wherein said build engine includes means for associating at least one of said individual audio tracks with a web page or an object on a web page and means for synchronizing multiple audio tracks.

29. The apparatus of claim 2, further including file size reduction means for reducing the total size of said run time files to less than approximately 50K and for reducing the total size of said the run time file of the first web page of said website to less than 25K.

30. The apparatus of claim 29, wherein said total size of said run time files is from approximately 12K to approximately 50K.

31. The apparatus of claim 29, wherein said elements include a web page, an object, a paragraph, or combinations thereof, and wherein said file size reduction means includes means for tracking multiple high water marks for said elements to minimize the size of said at least a portion of said database generated by said build tool.

32. The apparatus of claim 29, wherein said file size reduction means includes means for identify website specific run time files.

33. The apparatus of claim 2, wherein said website has a first web page, wherein said run time files includes a main run time file and main database information corresponding to said first web page, and further includes a run time routine for simultaneously obtaining from the Internet and displaying said first web page and processing said database corresponding to others of said web pages.

34. The apparatus of claim 33, wherein at least one of said elements includes one or more images, and wherein the run time files includes an image observer operable to permit said one or more images of said website to be downloaded simultaneously with the execution of said run time files.

35. The apparatus of claim 2, wherein the build engine includes dynamic resizing means operable to redefine a size of a web page upon being display.

36. The apparatus of claim 35, wherein the dynamic resizing apparatus can be invoked in real time during the build process when a new web site file is opened, when the web page size of the existing web site is changed, or when the web page is zoomed to a different size.

37. An apparatus for producing Internet websites having one or more web pages on and for a computer having a browser and a virtual machine capable of generating a display, said apparatus comprising:

- (a) an interface configured for building a website through control of website elements, said interface being operable through the browser on the computer to:
  - present a viewable menu of a user selectable panel of settings, accept a plurality of settings from said user selectable panel of settings to form an assembly of settings, and
  - generate the display in accordance with said assembly of settings contemporaneously with the acceptance thereof, at least one of said user selectable settings of said panel of settings being operable to generate said display through commands to said virtual machine;
- (b) an internal database associated with said interface for storing information representative of one or more of said assembly of settings for controlling elements of the website; and
- (c) a build tool to construct one or more web pages of said website having:
  - an external database containing data corresponding to said information stored in said internal database, and
  - one or more run time files,

## US 6,546,397 B1

69

where said run time files utilize information stored in said external database to generate virtual machine commands for the display of at least a portion of said one or more web pages.

38. The apparatus of claim 37, wherein said user selectable panel of settings are selected from the group consisting of construction, manipulation, animation and transition of elements.

39. An apparatus to allow users to produce Internet websites on and for computers having a browser and a virtual machine capable of generating displays, said apparatus comprising:

- (a) means for presenting a viewable menu of a user selectable panel of settings to describe elements on a website, said panel of settings being presented through a browser on a computer adapted to accept one or more of said selectable settings in said panel as inputs

70

therefrom, and where at least one of said user selectable settings in said panel corresponds to commands to said virtual machine;

- (b) means for generating a display in accordance with one or more user selected settings substantially contemporaneously with the selection thereof;
- (c) means for storing information in a database, said information being representative of a plurality of said one or more user selected settings; and
- (d) means for building one or more web pages for generating said website from at least a portion of said database and at least one run time file, where said at least one run time file utilizes information stored in said database to generate virtual machine commands for the display of at least a portion of said one or more web pages.

\* \* \* \* \*



US007594168B2

(12) **United States Patent**  
**Rempell**

(10) **Patent No.:** **US 7,594,168 B2**  
(45) **Date of Patent:** **Sep. 22, 2009**

(54) **BROWSER BASED WEB SITE GENERATION  
TOOL AND RUN TIME ENGINE**

(75) Inventor: **Steven H. Rempell**, Novato, CA (US)

(73) Assignee: **Akira Technologies, Inc.**

(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1100 days.

(21) Appl. No.: **10/351,182**

(22) Filed: **Jan. 24, 2003**

(65) **Prior Publication Data**  
US 2004/0148307 A1 Jul. 29, 2004

**Related U.S. Application Data**  
(63) Continuation of application No. 09/454,061, filed on Dec. 2, 1999, now Pat. No. 6,546,397.

(51) **Int. Cl.**  
**G06F 17/00** (2006.01)

(52) **U.S. Cl.** ..... **715/234; 715/238; 715/762**

(58) **Field of Classification Search** ..... **715/234,**  
**715/238, 762**  
See application file for complete search history.

(56) **References Cited**  
**U.S. PATENT DOCUMENTS**

5,428,731 A 6/1995 Powers, III  
5,842,020 A 11/1998 Faustini  
5,870,767 A 2/1999 Kraft, IV  
6,026,433 A \* 2/2000 D'Arlach et al. .... 709/217  
6,081,263 A 6/2000 LeGall et al.  
6,083,276 A 7/2000 Davidson et al.  
6,148,311 A 11/2000 Wishnie et al.

6,262,729 B1 \* 7/2001 Marcos et al. .... 715/744  
6,313,835 B1 \* 11/2001 Gever et al. .... 715/846  
6,424,979 B1 \* 7/2002 Livingston et al. .... 715/206  
6,585,779 B1 \* 7/2003 Becker ..... 715/237  
6,675,382 B1 \* 1/2004 Foster ..... 717/177  
6,684,369 B1 \* 1/2004 Bernardo et al. .... 715/205  
7,127,501 B1 \* 10/2006 Beir et al. .... 709/219  
7,152,207 B1 \* 12/2006 Underwood et al. .... 715/207  
2001/0011287 A1 \* 8/2001 Goto et al. .... 707/513  
2001/0042083 A1 \* 11/2001 Saito et al. .... 707/517  
2003/0058277 A1 \* 3/2003 Bowman-Amuah ..... 345/765  
2005/0198087 A1 \* 9/2005 Bremers ..... 707/204  
2005/0223320 A1 \* 10/2005 Brintzenhofe et al. .... 715/517  
2009/0094327 A1 \* 4/2009 Shuster et al. .... 709/203

**OTHER PUBLICATIONS**

Tyler, Denise, Microsoft Frontpage 98, Nov. 1997, pp. 276-281, 321-327, 558-561.\*  
"Photo Album Applet Installation and Customization Instructions" © 1997 AgenX Corp, <http://pages.prodigy.net/larzman/applets/phalbum.html>.\*

\* cited by examiner

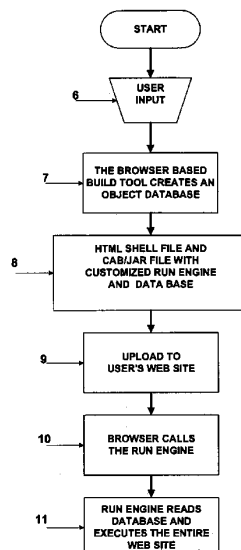
*Primary Examiner*—Adam M Queler

(74) *Attorney, Agent, or Firm*—Steven R. Vosen

(57) **ABSTRACT**

A method and apparatus for designing and building a web page. The apparatus includes a browser based build engine including build tools and a user interface. The build tools are operable to construct a single run time file and an associated database that describe, and when executed, produce the web page. The user interface includes a build frame and a panel. The build frame is operable to receive user input and present a WYSIWIG representation of the web page. The panel includes one or more menus for controlling the form of content to be placed on the web page.

**6 Claims, 68 Drawing Sheets**



U.S. Patent

Sep. 22, 2009

Sheet 1 of 68

US 7,594,168 B2

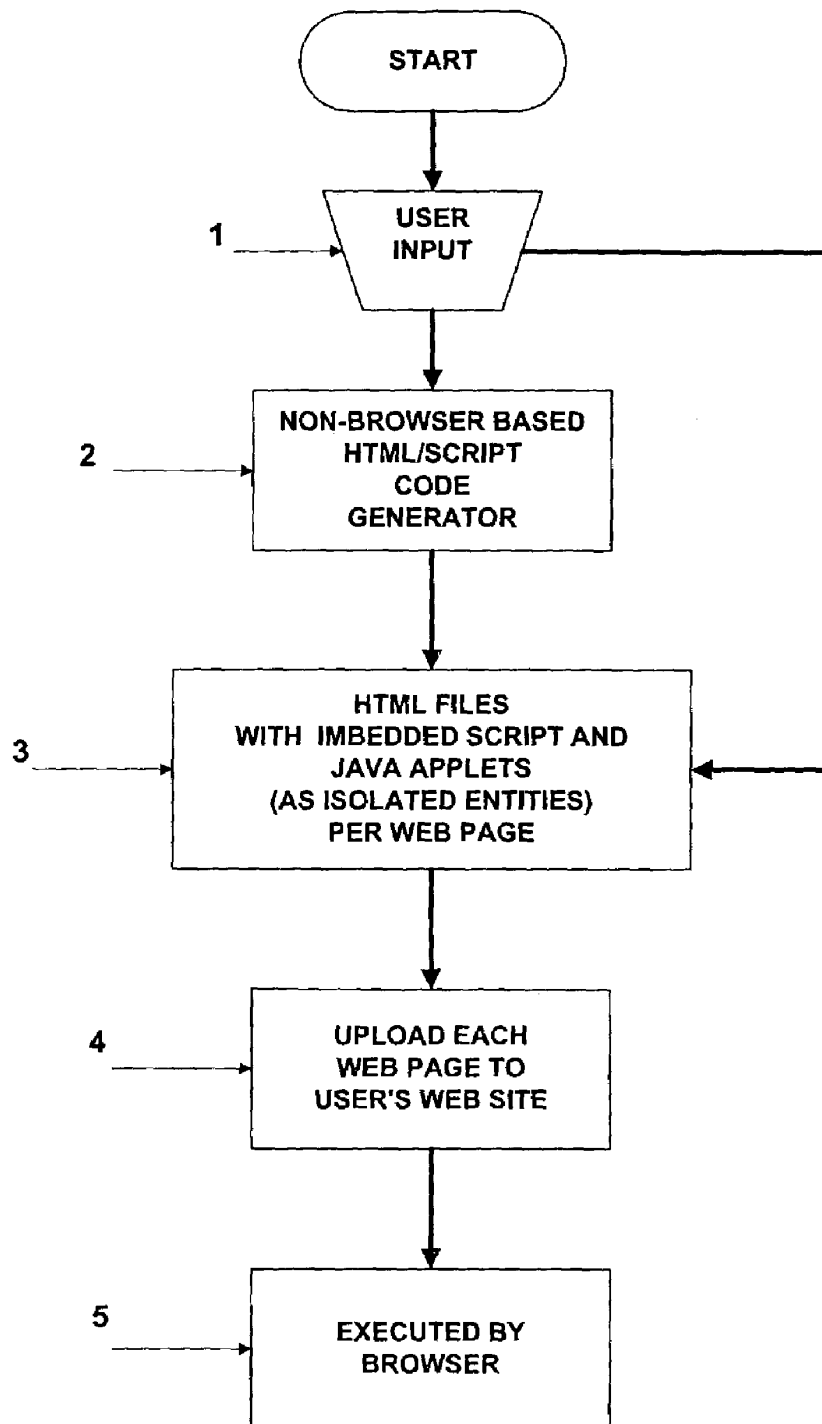


Fig. 1

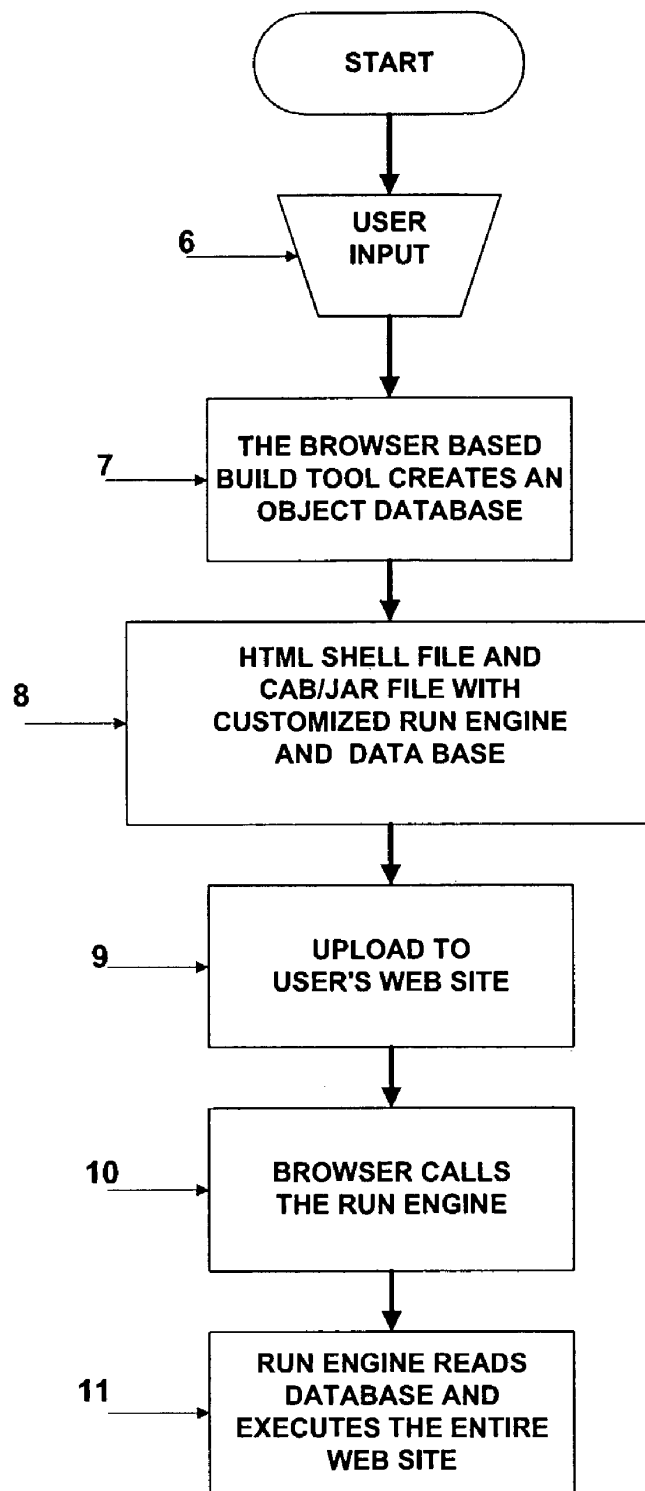
PRIOR ART

U.S. Patent

Sep. 22, 2009

Sheet 2 of 68

US 7,594,168 B2



**Fig. 2**

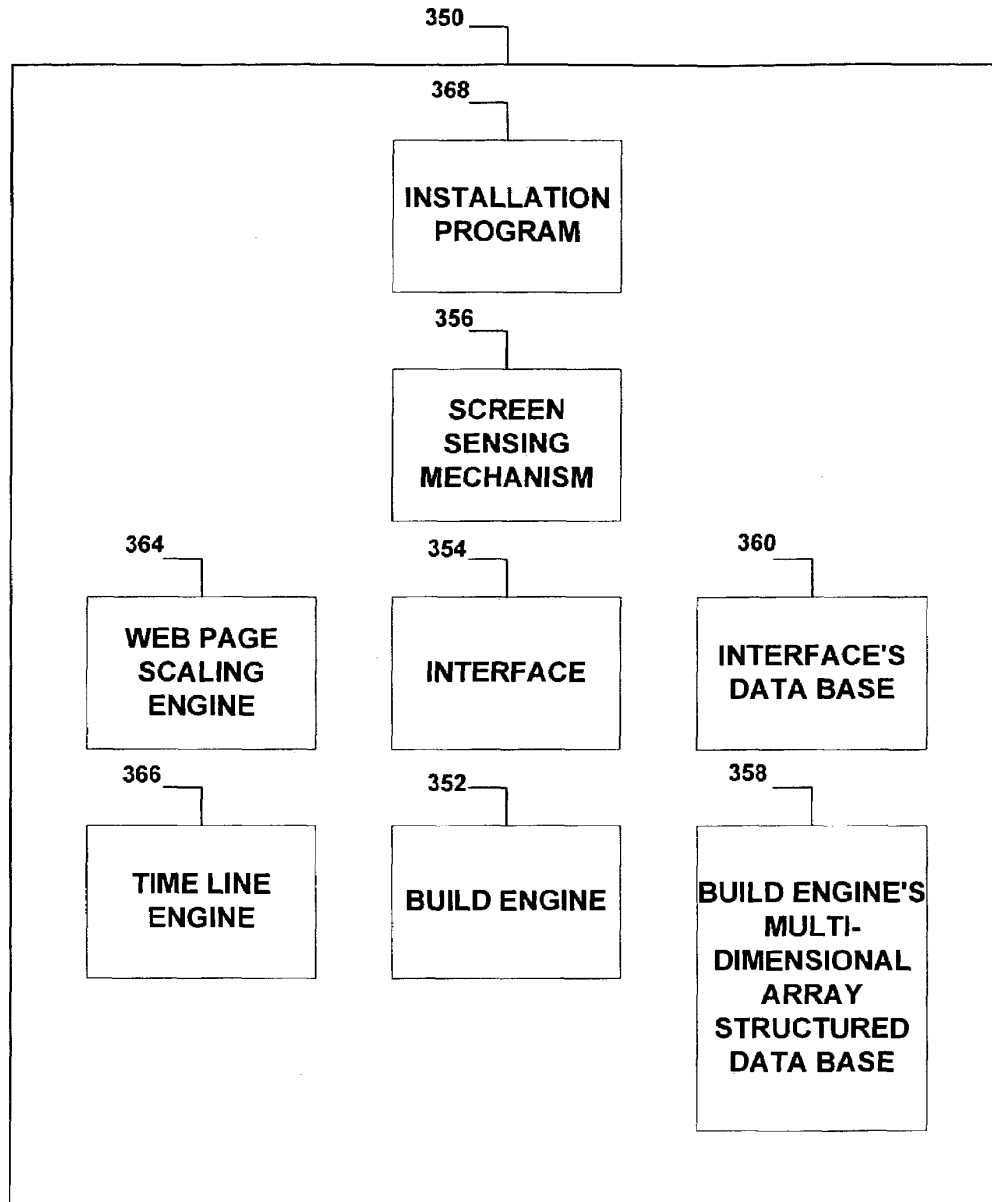


U.S. Patent

Sep. 22, 2009

Sheet 3 of 68

US 7,594,168 B2



**Fig. 3a**

## BUILD TOOL COMPONENTS

U.S. Patent

Sep. 22, 2009

Sheet 4 of 68

US 7,594,168 B2

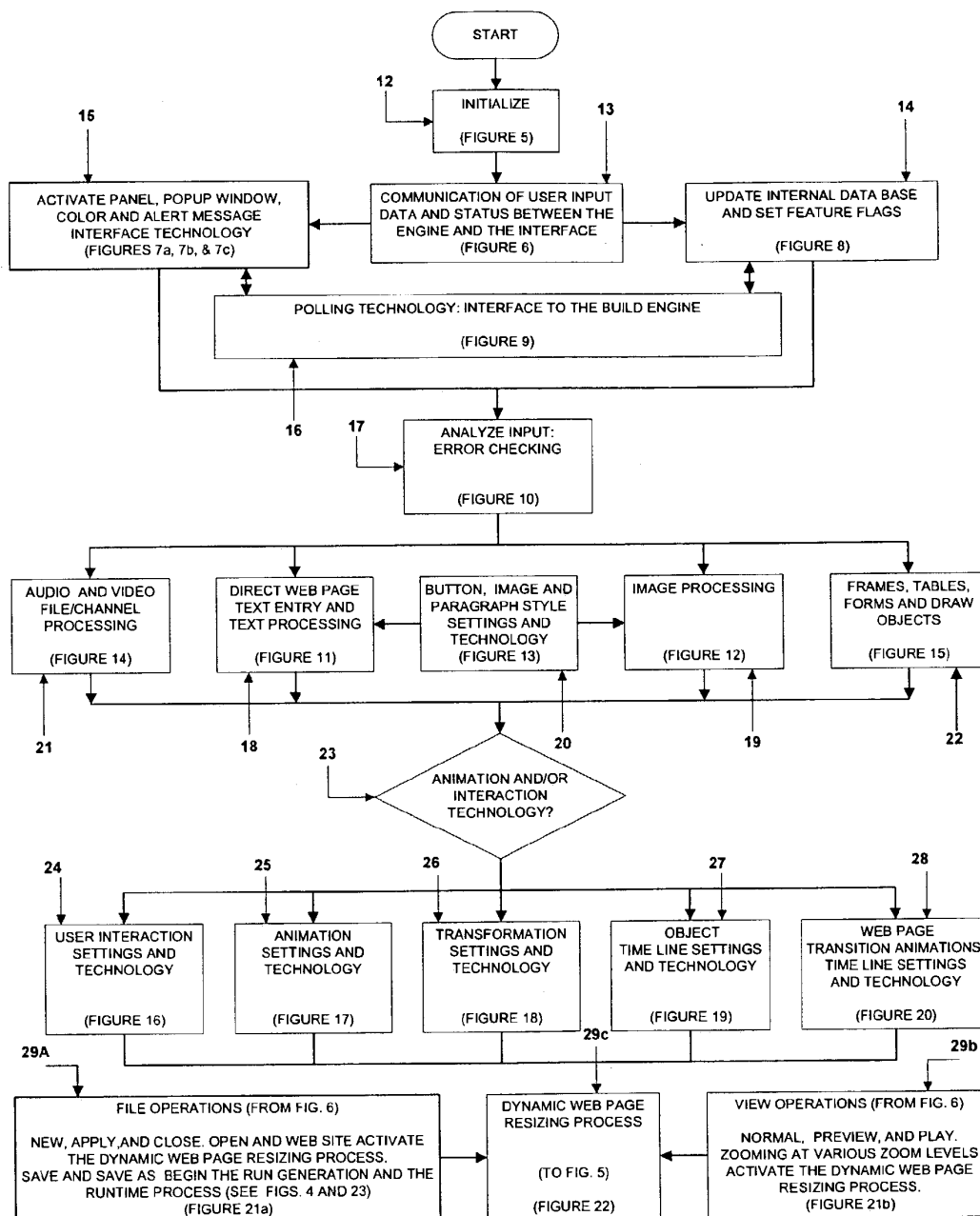


Fig. 3b

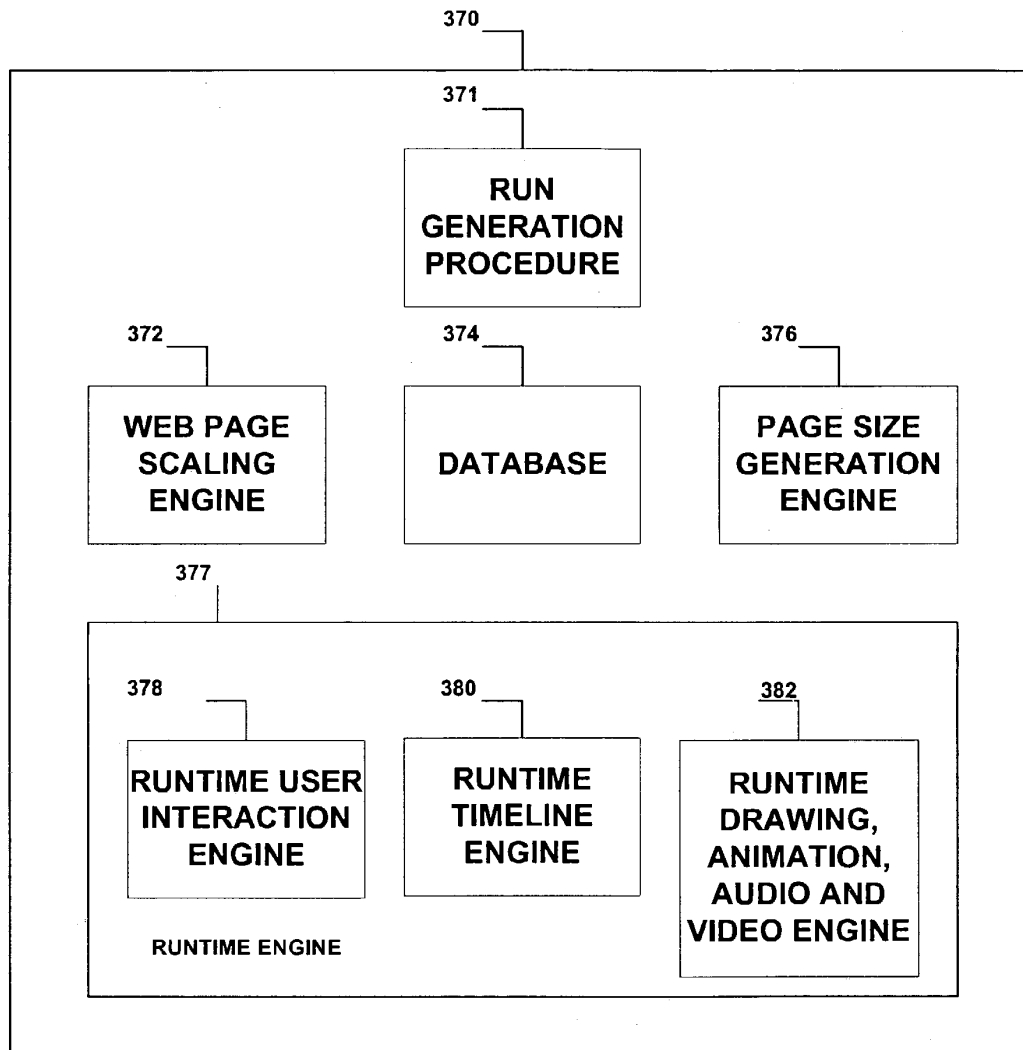
## THE BUILD TOOL &amp; BUILD PROCESS

U.S. Patent

Sep. 22, 2009

Sheet 5 of 68

US 7,594,168 B2



**Fig. 4a**

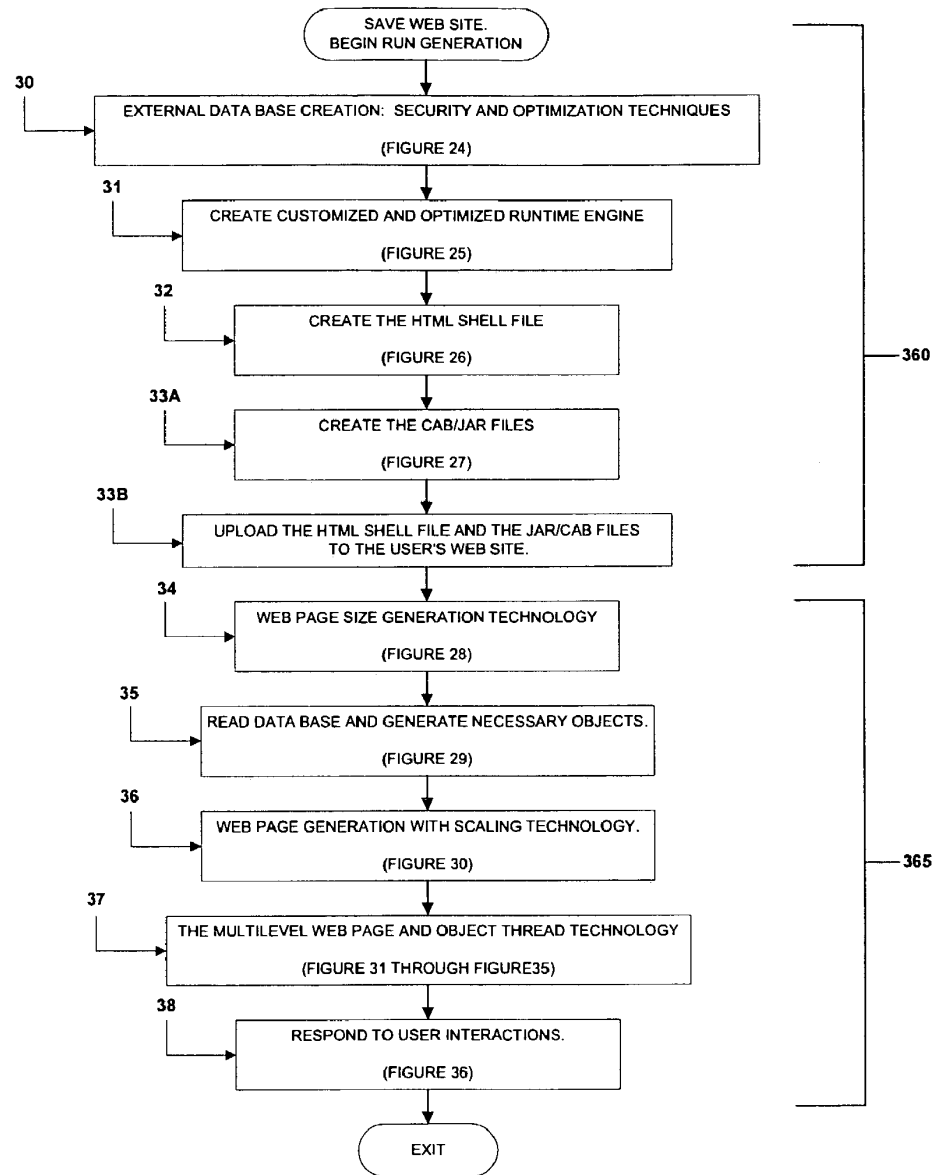
## **RUN GENERATION AND RUNTIME COMPONENTS**

U.S. Patent

Sep. 22, 2009

Sheet 6 of 68

US 7,594,168 B2

*Fig. 4b***RUN GENERATION & THE RUNTIME PROCESS**

U.S. Patent

Sep. 22, 2009

Sheet 7 of 68

US 7,594,168 B2

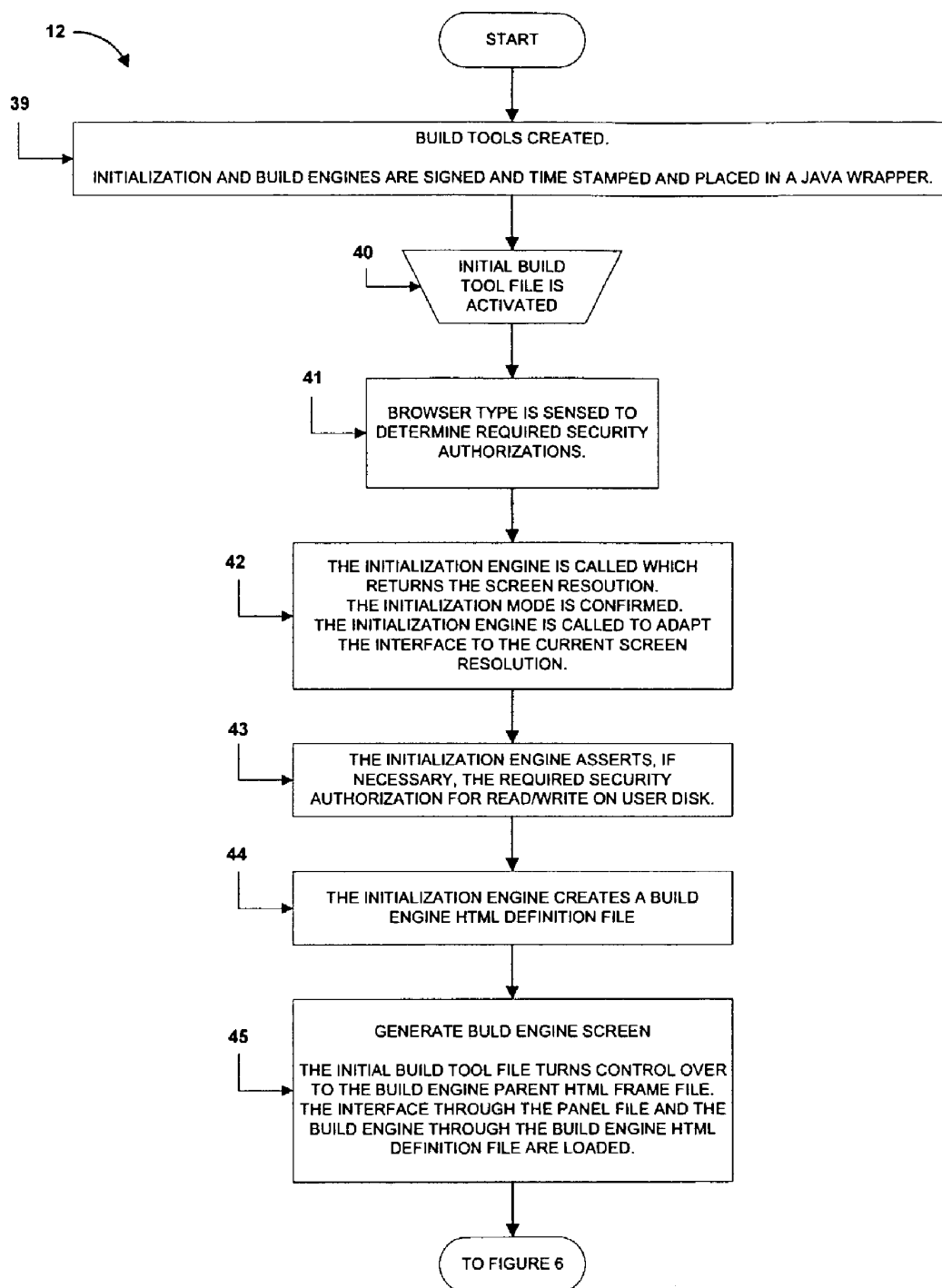


Fig . 5

## INITIALIZATION

U.S. Patent

Sep. 22, 2009

Sheet 8 of 68

US 7,594,168 B2

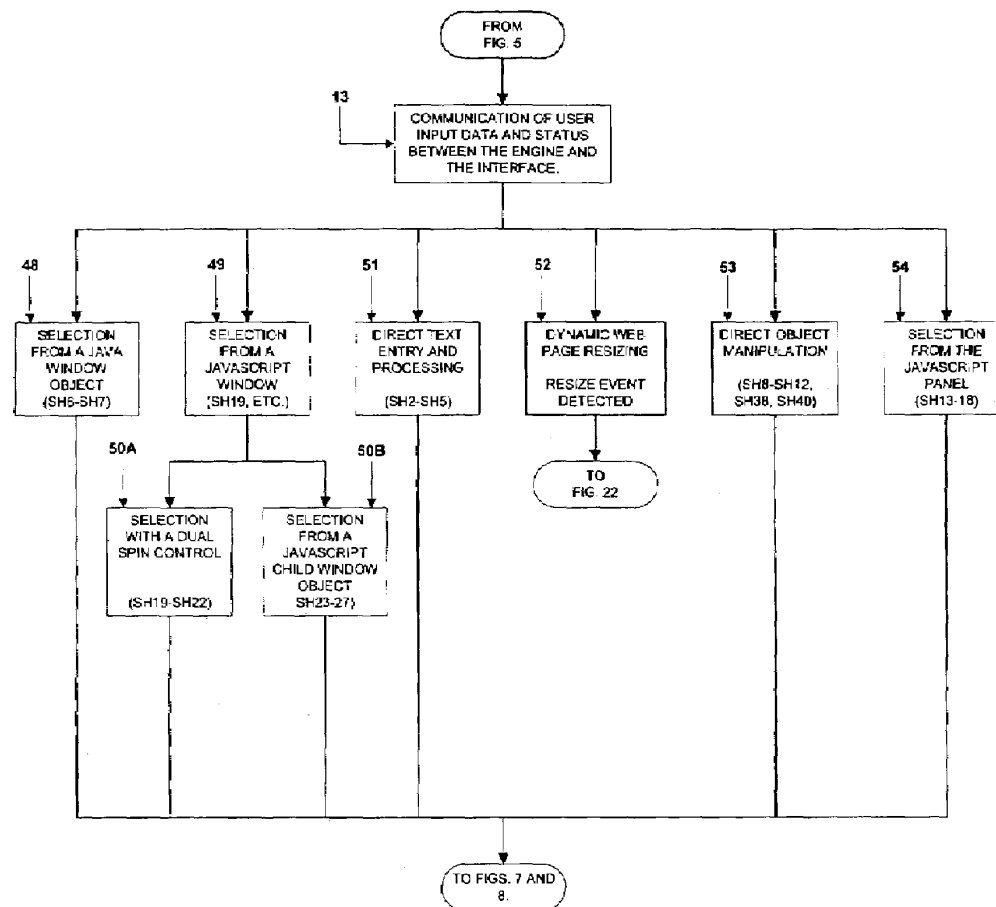


Fig. 6

**COMMUNICATION OF USER INPUT DATA AND  
STATUS BETWEEN THE ENGINE AND THE  
INTERFACE.**

U.S. Patent

Sep. 22, 2009

Sheet 9 of 68

US 7,594,168 B2

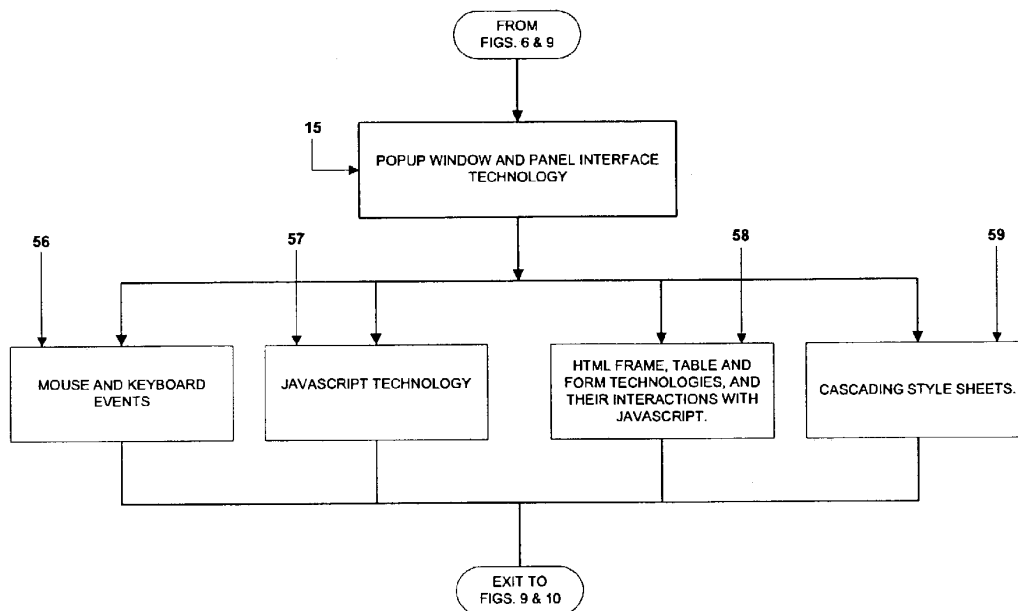


Fig. 7a

**POPUP WINDOW AND PANEL INTERFACE  
AND COLOR TECHNOLOGY**

U.S. Patent

Sep. 22, 2009

Sheet 10 of 68

US 7,594,168 B2

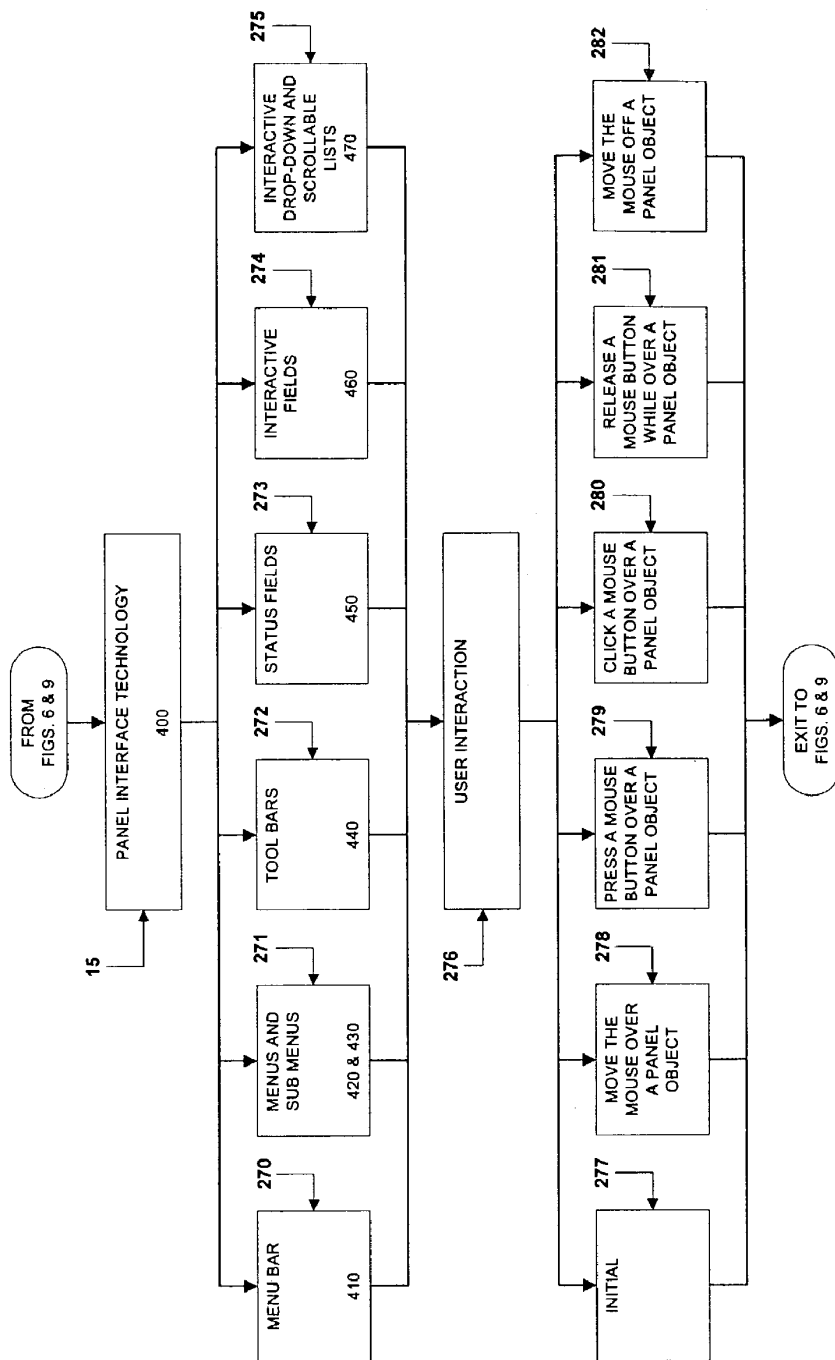


Fig. 7b

# IMPLEMENTATION OF PANEL INTERFACE OBJECTS

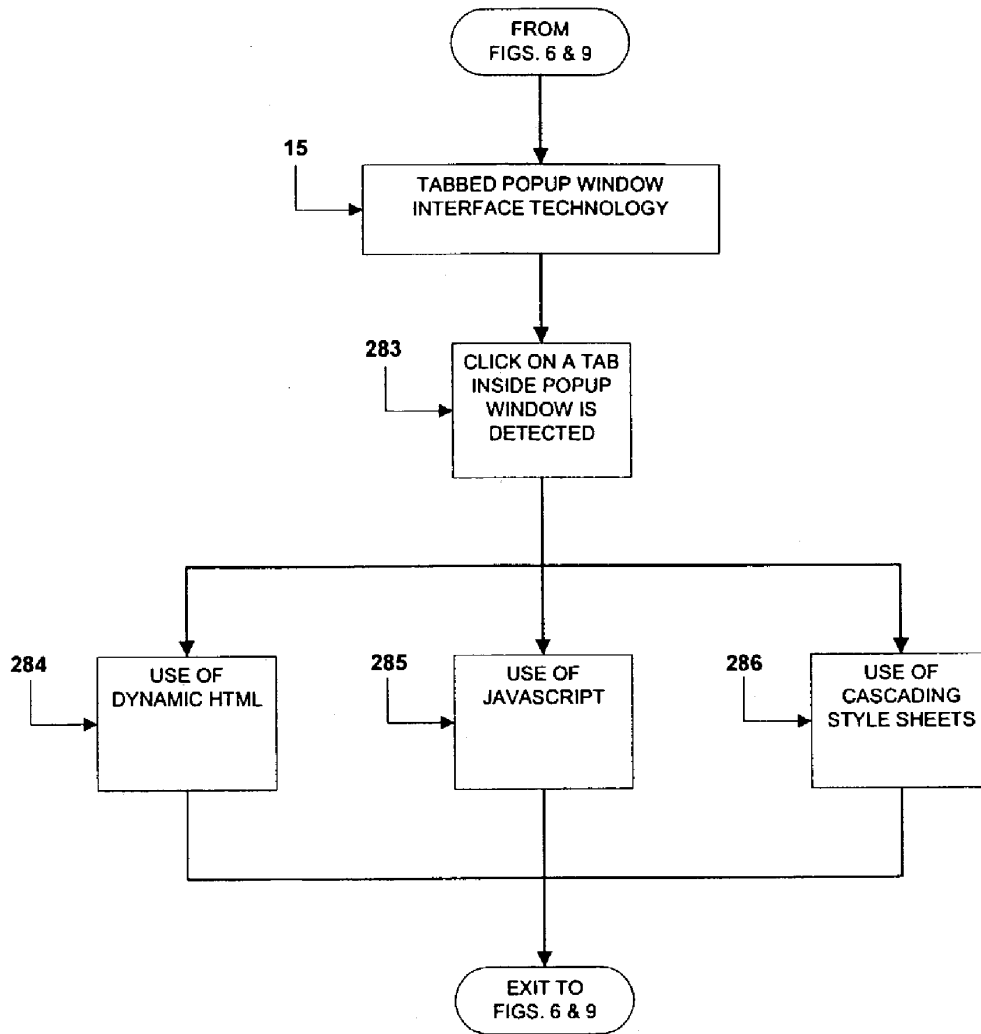


U.S. Patent

Sep. 22, 2009

Sheet 11 of 68

US 7,594,168 B2



*Fig. 7c*

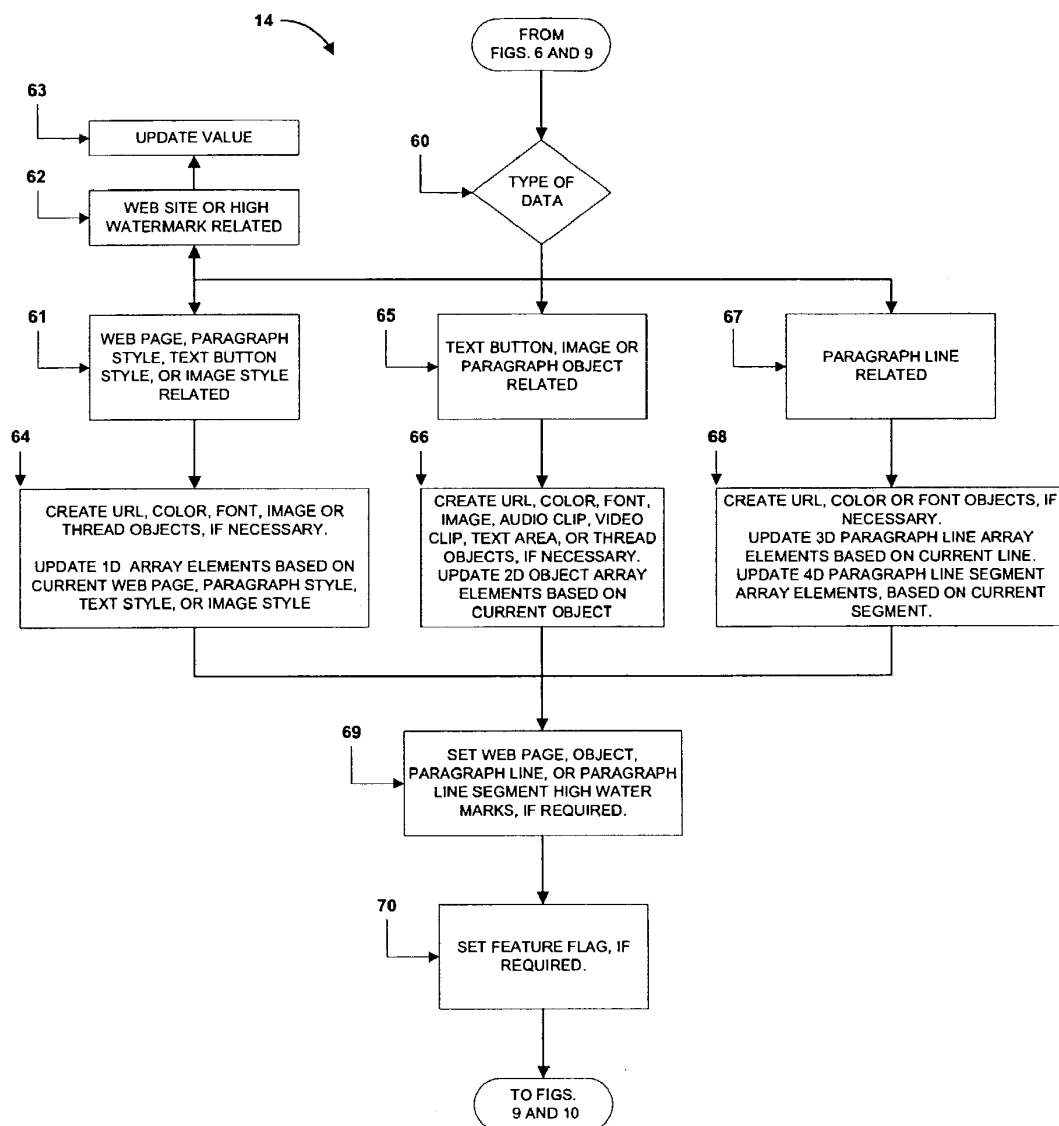
## IMPLEMENTATION OF TABBED POPUP WINDOWS

U.S. Patent

Sep. 22, 2009

Sheet 12 of 68

US 7,594,168 B2

*Fig. 8***UPDATE INTERNAL DATA BASE AND SET FEATURE FLAGS**

U.S. Patent

Sep. 22, 2009

Sheet 13 of 68

US 7,594,168 B2

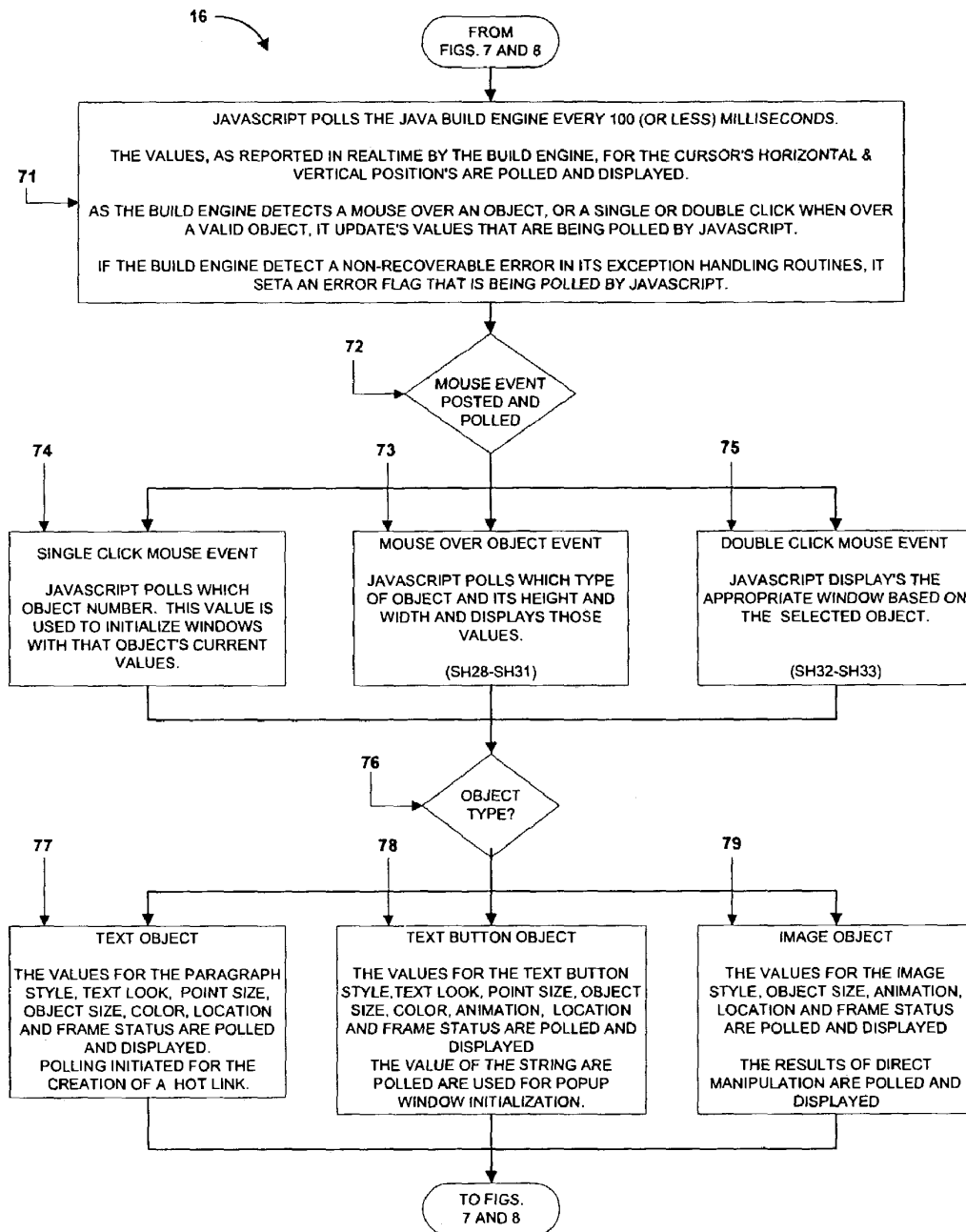


Fig. 9

## POLLING METHODS

U.S. Patent

Sep. 22, 2009

Sheet 14 of 68

US 7,594,168 B2

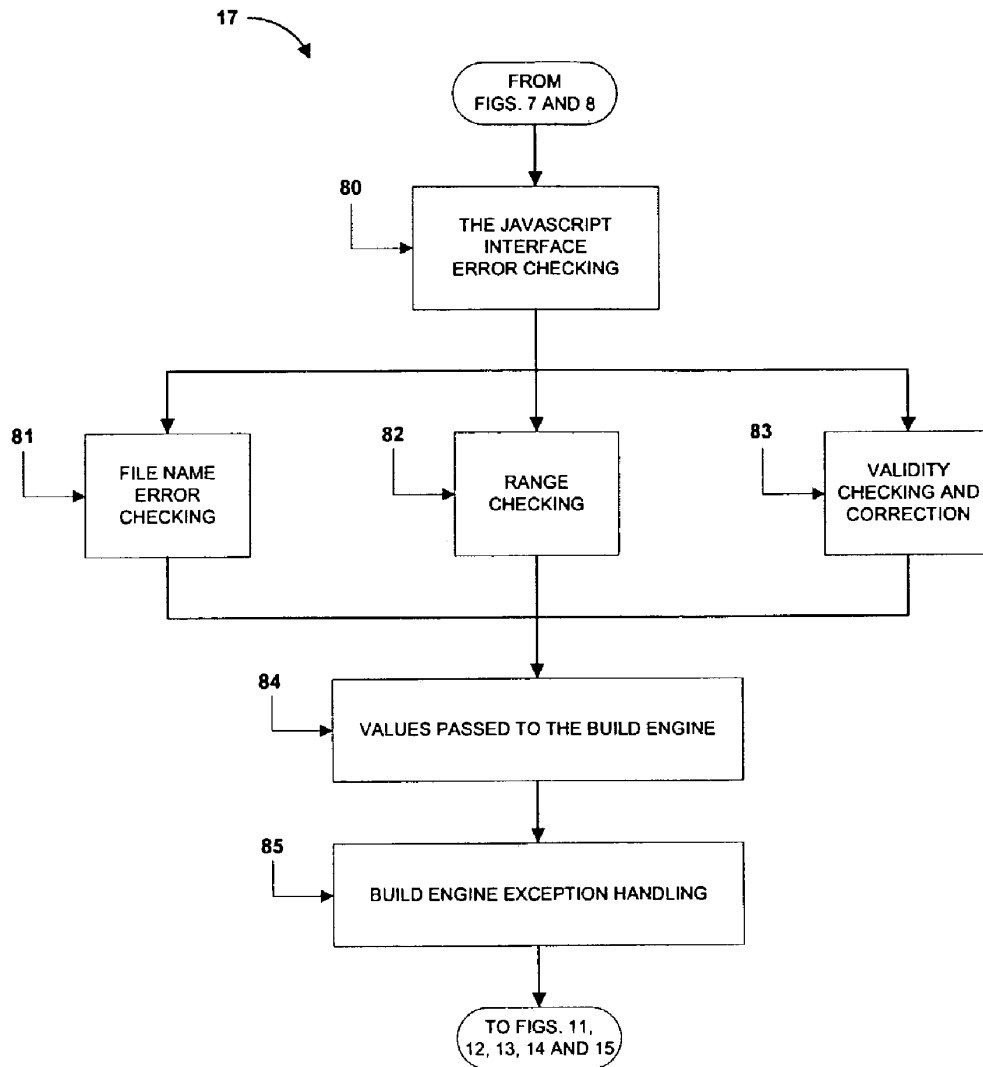


Fig. 10

**ANALYZE INPUT: ERROR CHECKING**

U.S. Patent

Sep. 22, 2009

Sheet 15 of 68

US 7,594,168 B2

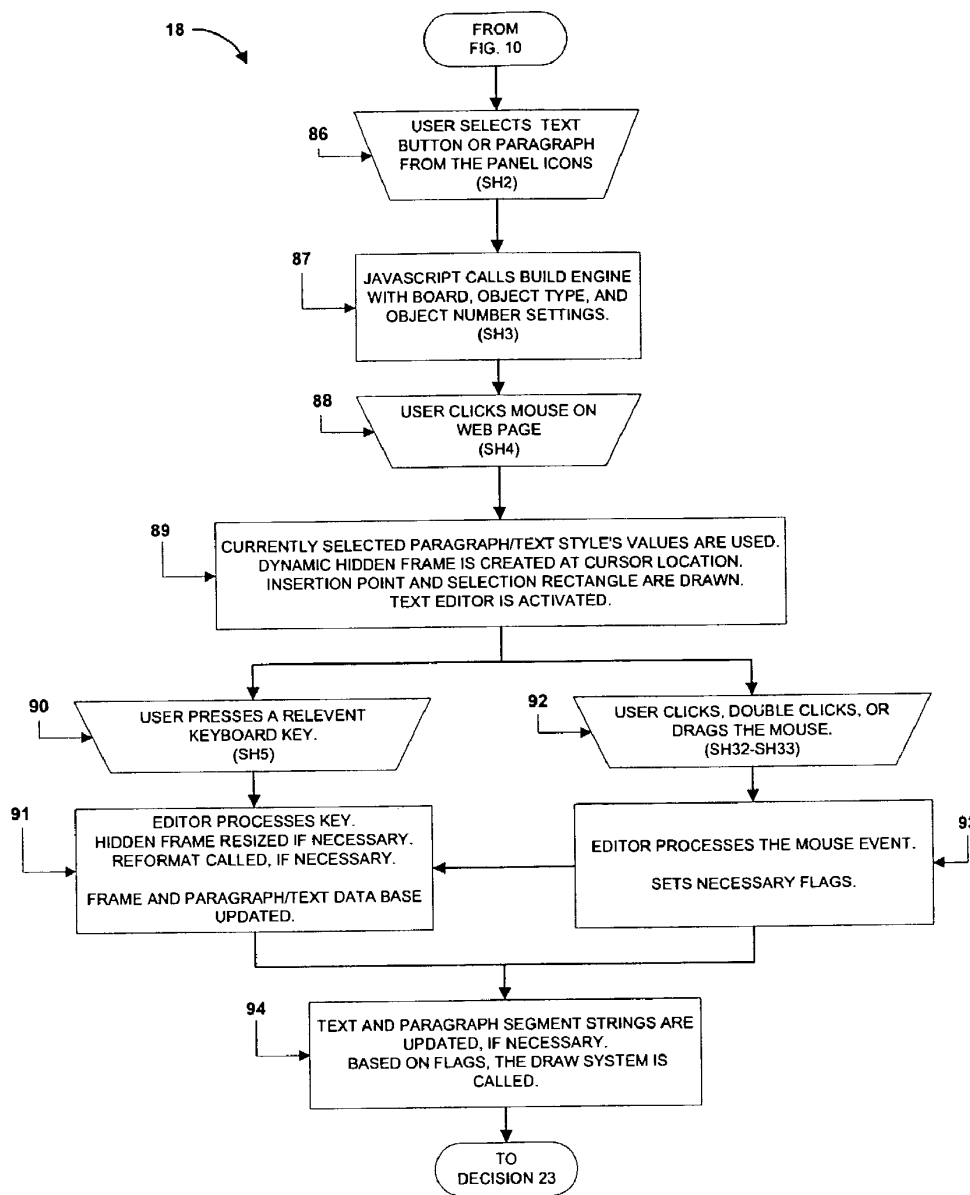


Fig. 11

## DIRECT WEB PAGE DATA ENTRY AND TEXT PROCESSING

U.S. Patent

Sep. 22, 2009

Sheet 16 of 68

US 7,594,168 B2

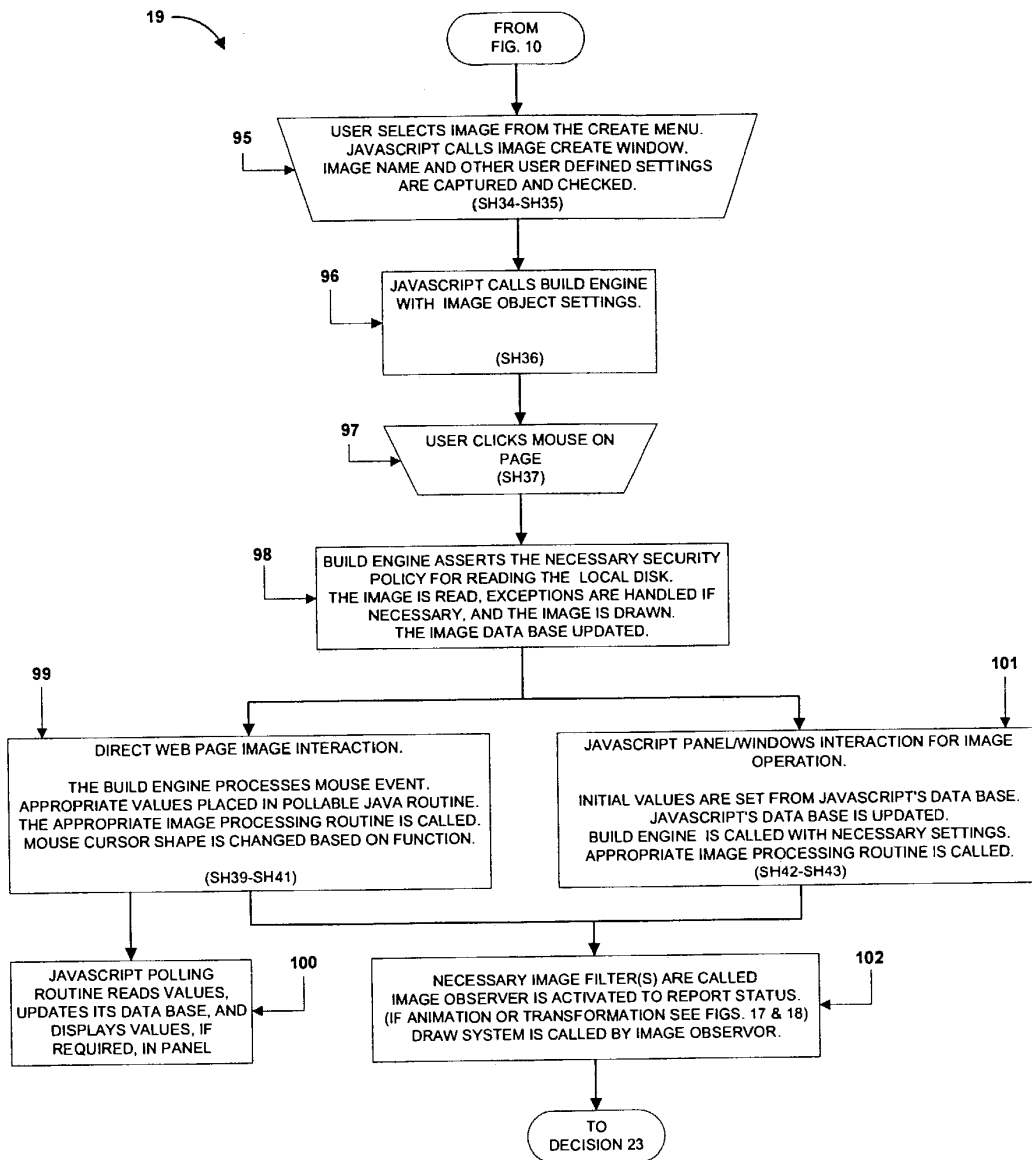


Fig. 12

## IMAGE PROCESSING

U.S. Patent

Sep. 22, 2009

Sheet 17 of 68

US 7,594,168 B2

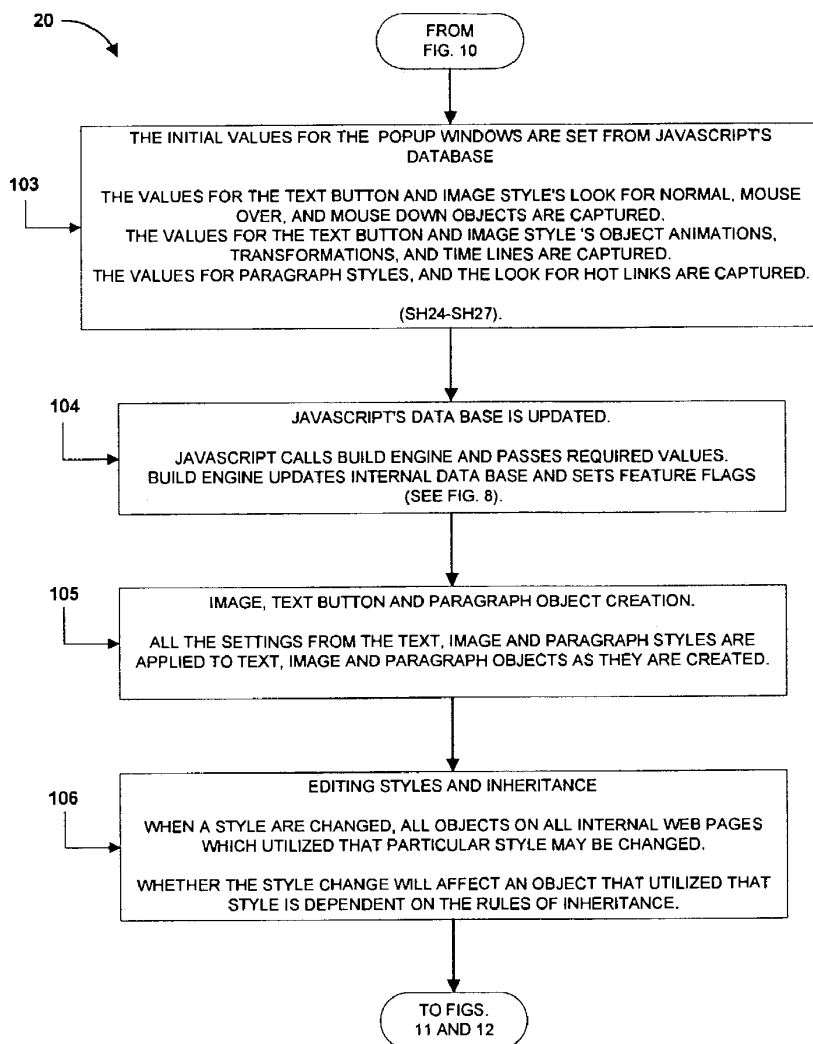


Fig. 13

**BUTTON, IMAGE AND PARAGRAPH STYLE SETTINGS  
AND TECHNOLOGY**

U.S. Patent

Sep. 22, 2009

Sheet 18 of 68

US 7,594,168 B2

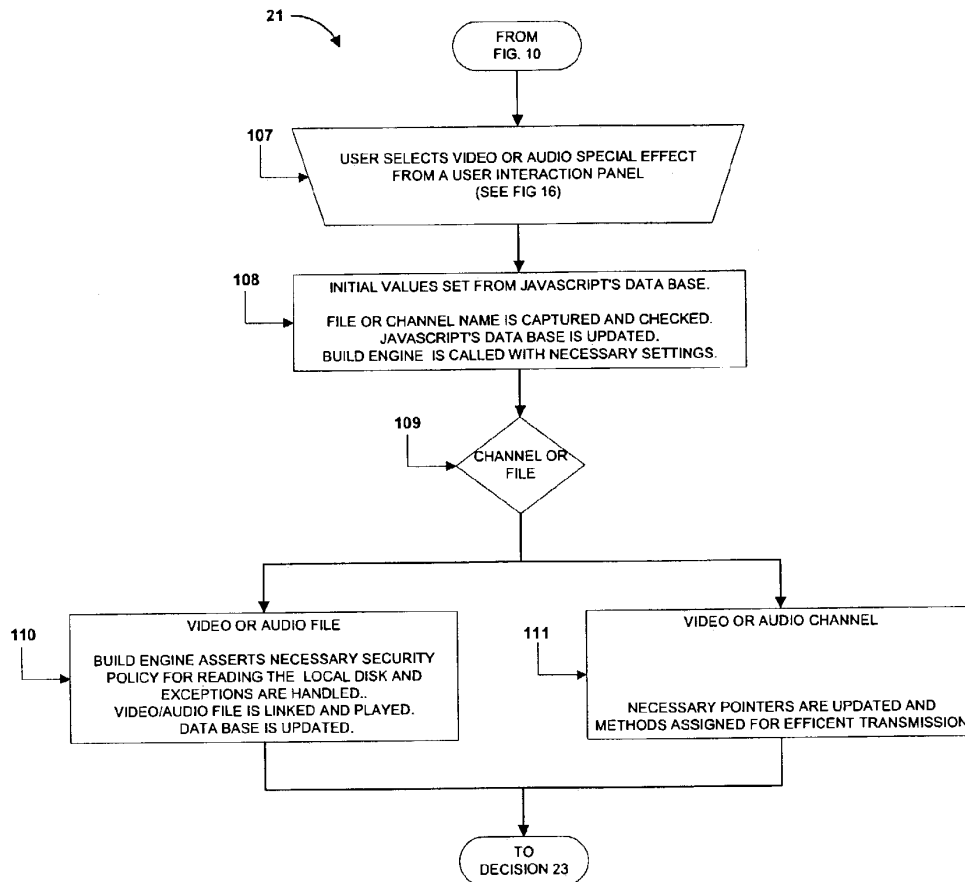


Fig. 14

## VIDEO AND AUDIO FILE/CHANNEL PROCESSING



U.S. Patent

Sep. 22, 2009

Sheet 19 of 68

US 7,594,168 B2

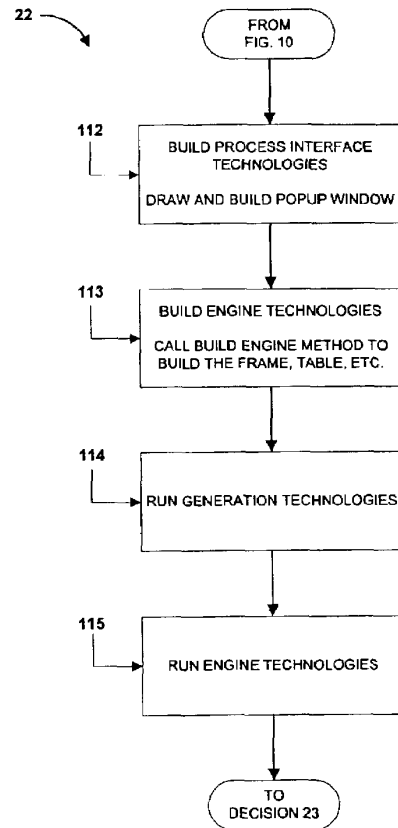


Fig. 15

FRAMES, TABLES, FORMS AND DRAW OBJECTS

U.S. Patent

Sep. 22, 2009

Sheet 20 of 68

US 7,594,168 B2

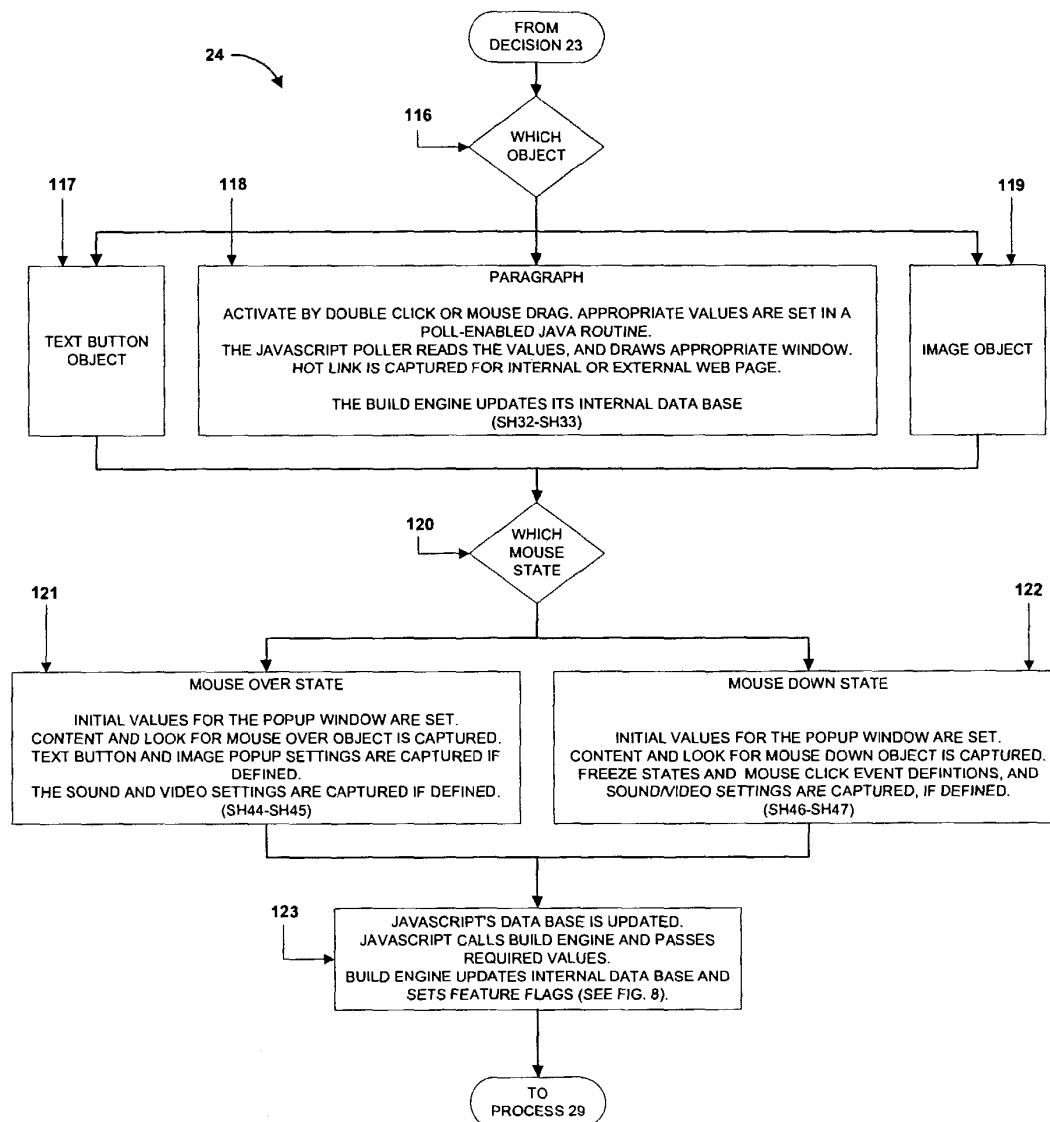


Fig. 16

USER INTERACTION SETTINGS AND TECHNOLOGY

U.S. Patent

Sep. 22, 2009

Sheet 21 of 68

US 7,594,168 B2

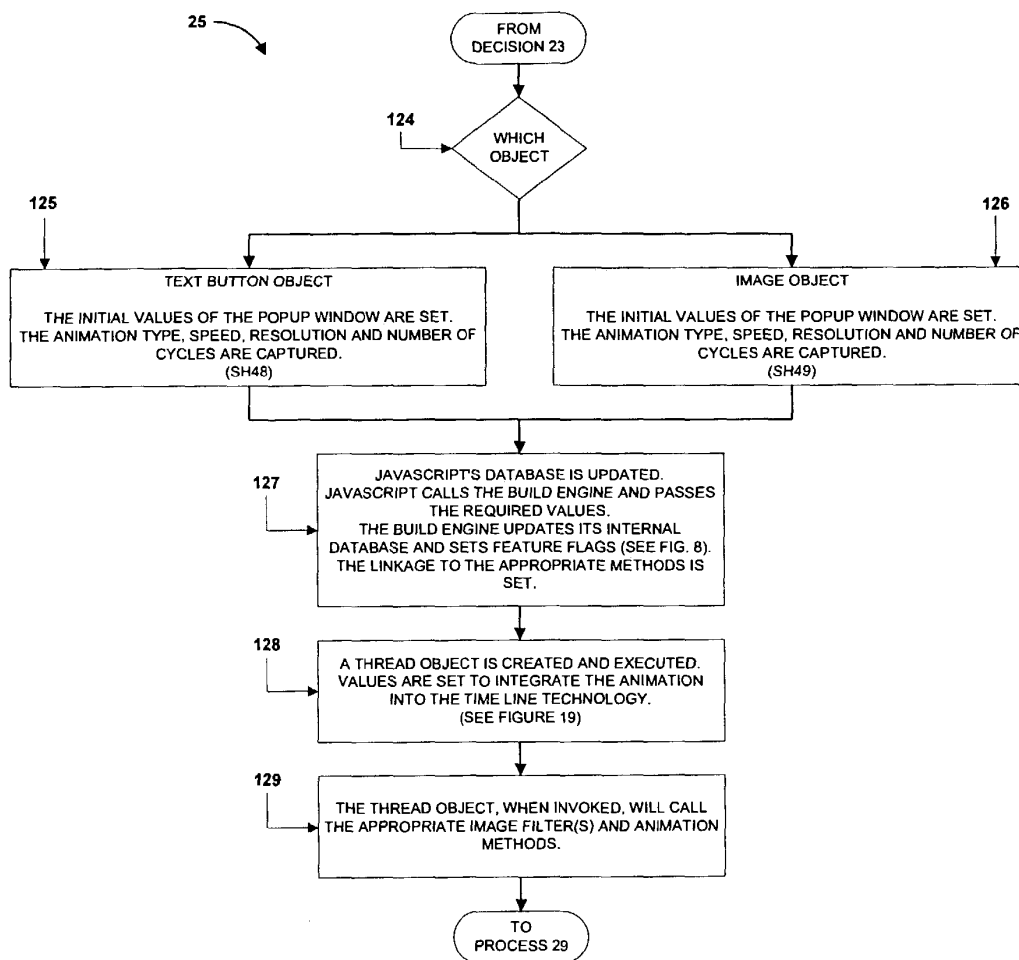


Fig. 17

ANIMATION SETTINGS AND TECHNOLOGY
-----------------------------------

U.S. Patent

Sep. 22, 2009

Sheet 22 of 68

US 7,594,168 B2

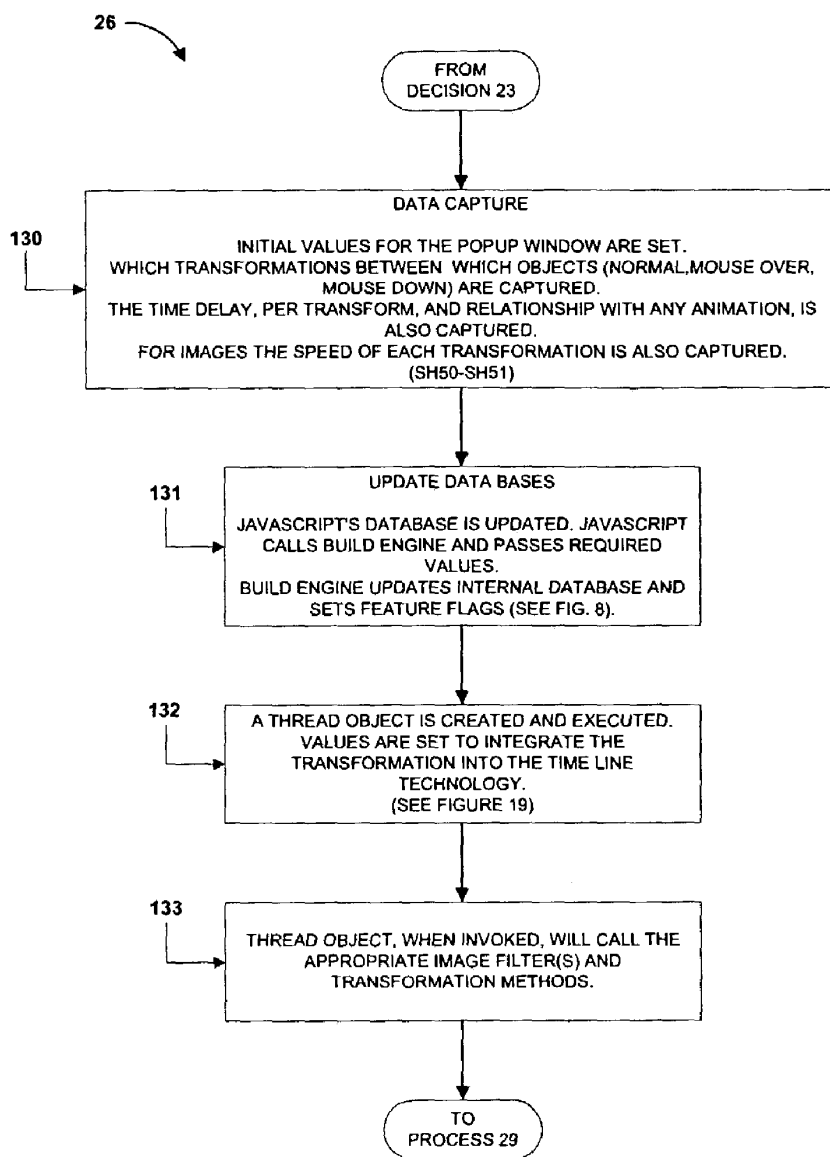


Fig. 18

**TRANSFORMATION SETTINGS AND TECHNOLOGY**

U.S. Patent

Sep. 22, 2009

Sheet 23 of 68

US 7,594,168 B2

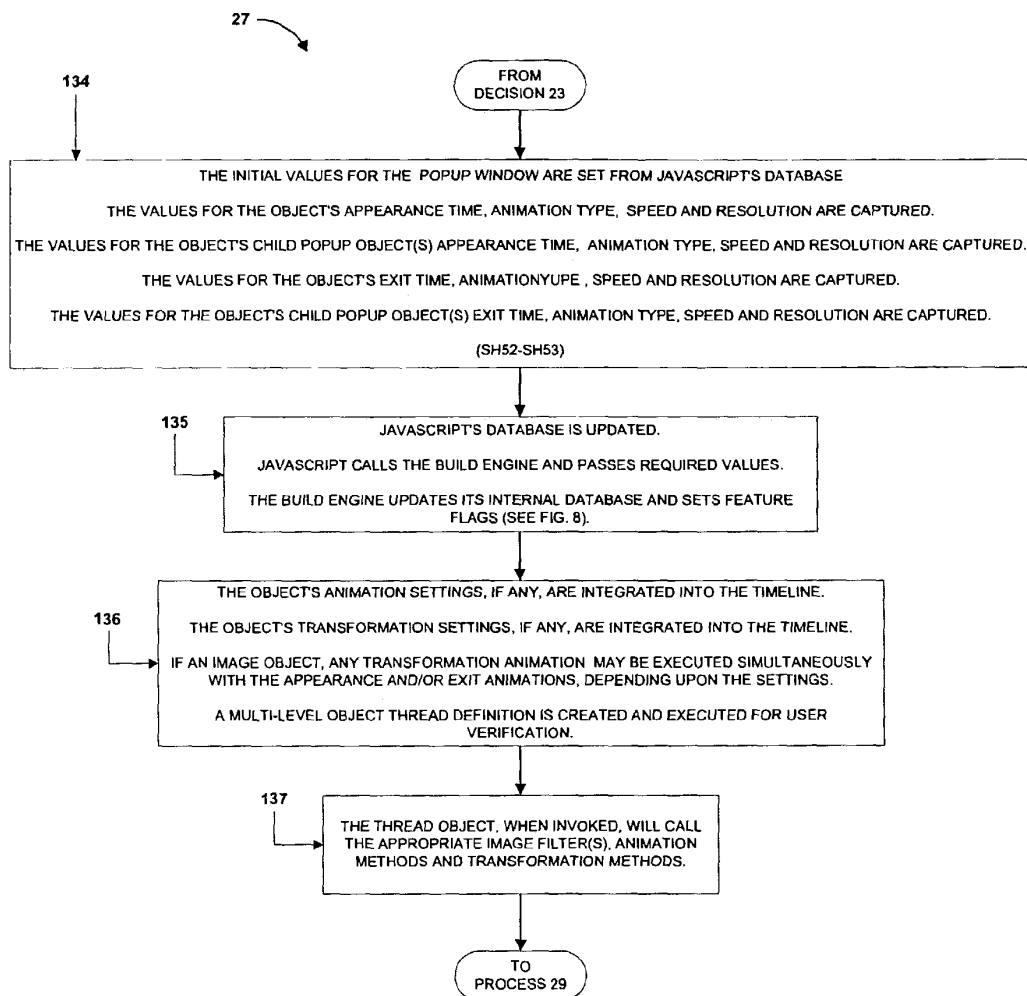


Fig. 19

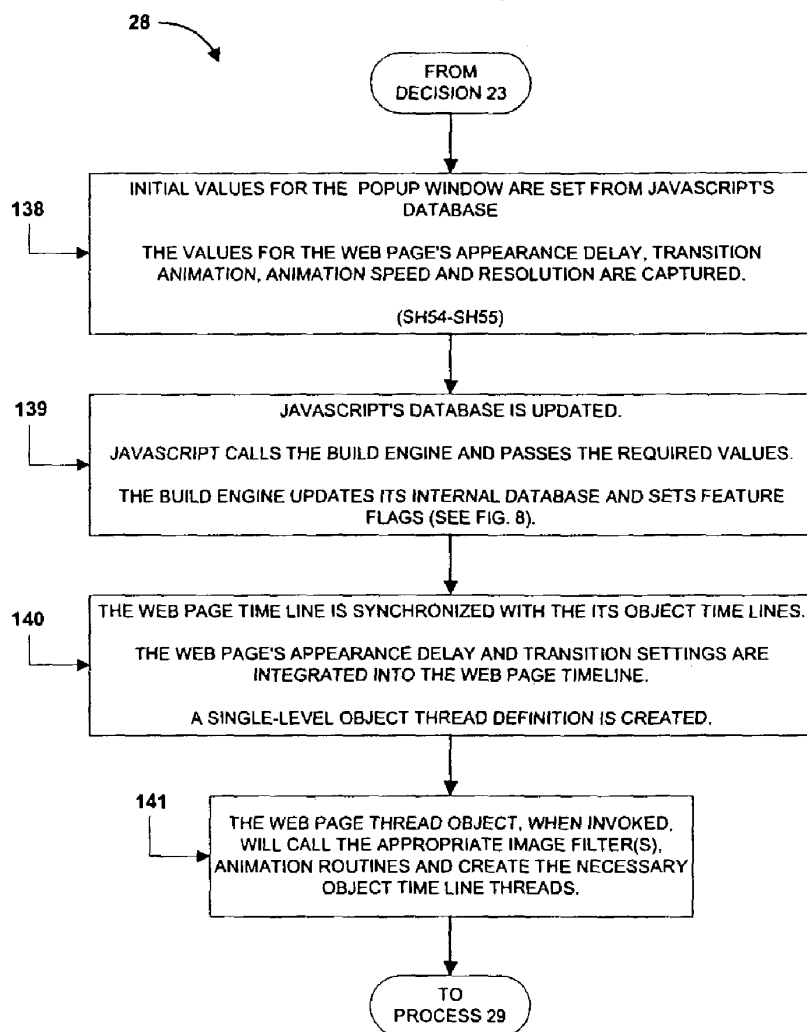
OBJECT TIME LINES AND TECHNOLOGY

U.S. Patent

Sep. 22, 2009

Sheet 24 of 68

US 7,594,168 B2

*Fig. 20*

**WEB PAGE TRANSITION ANIMATIONS, TIME LINE  
SETTINGS AND TECHNOLOGY**

U.S. Patent

Sep. 22, 2009

Sheet 25 of 68

US 7,594,168 B2

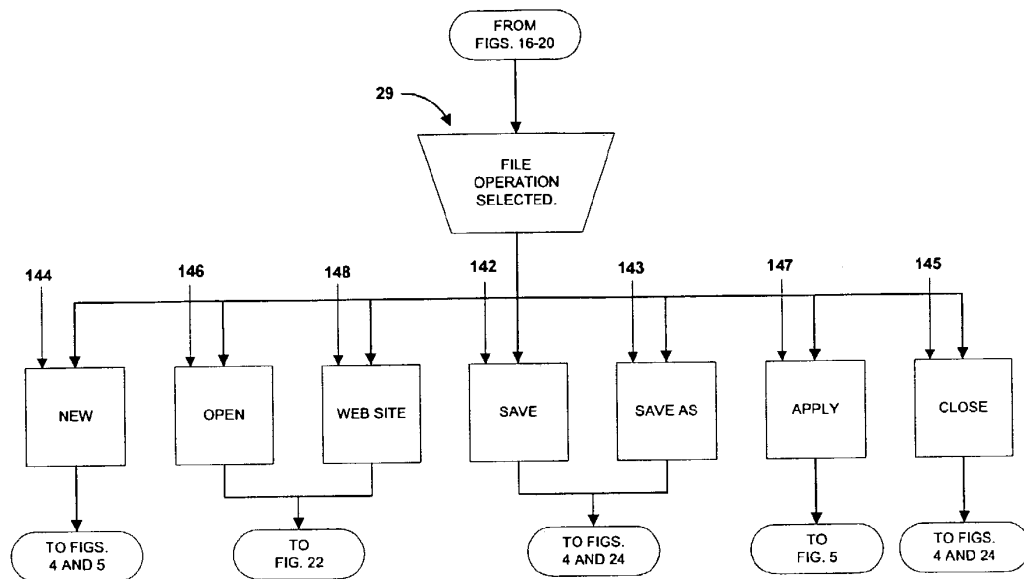


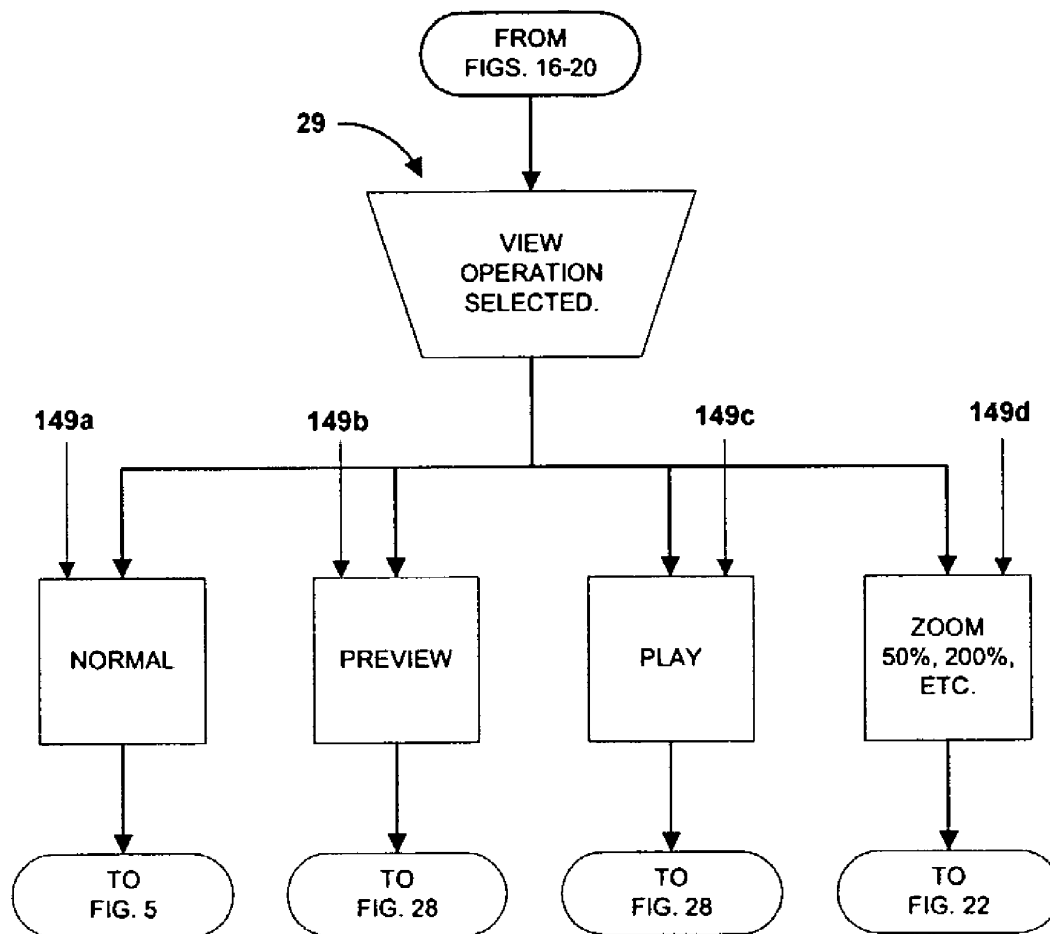
fig. 21a

U.S. Patent

Sep. 22, 2009

Sheet 26 of 68

US 7,594,168 B2



*fig. 21b*



U.S. Patent

Sep. 22, 2009

Sheet 27 of 68

US 7,594,168 B2

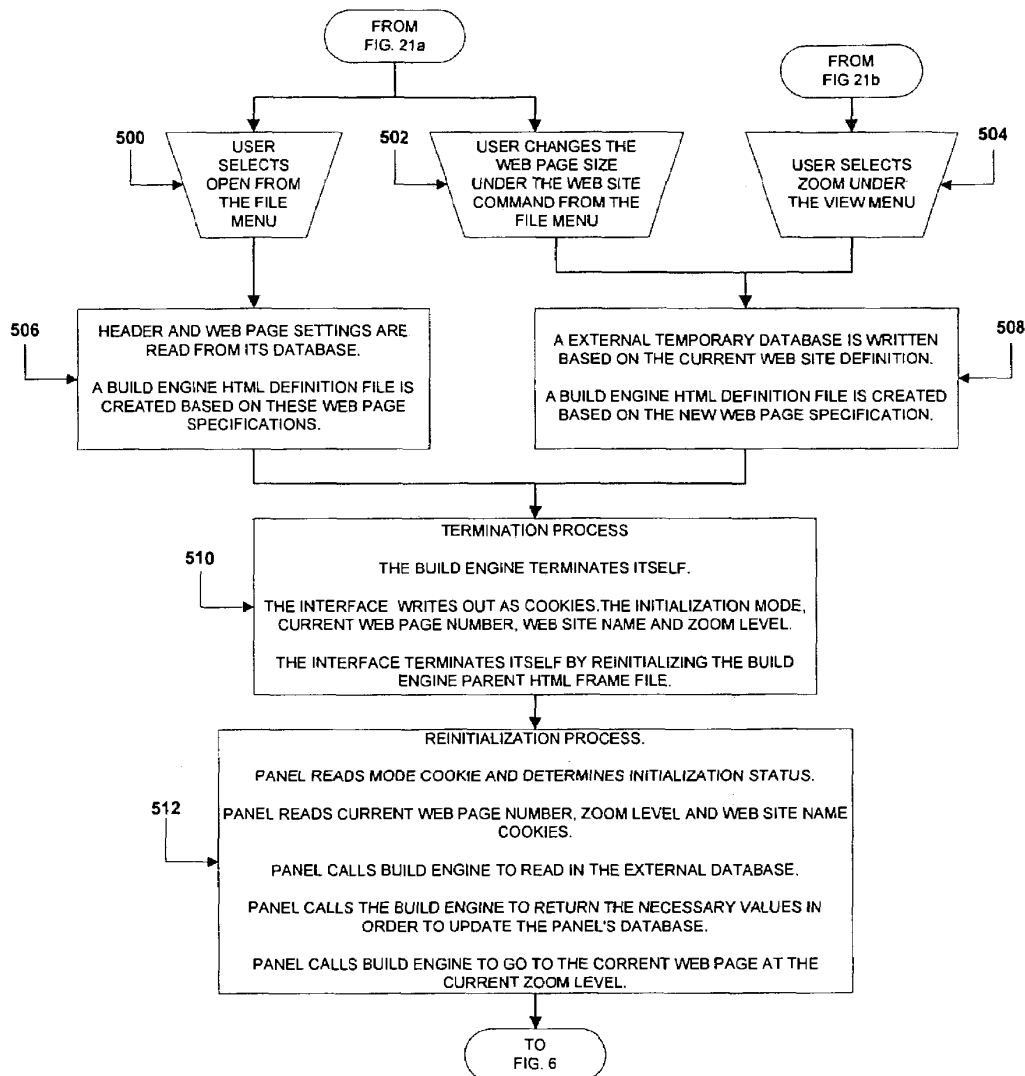
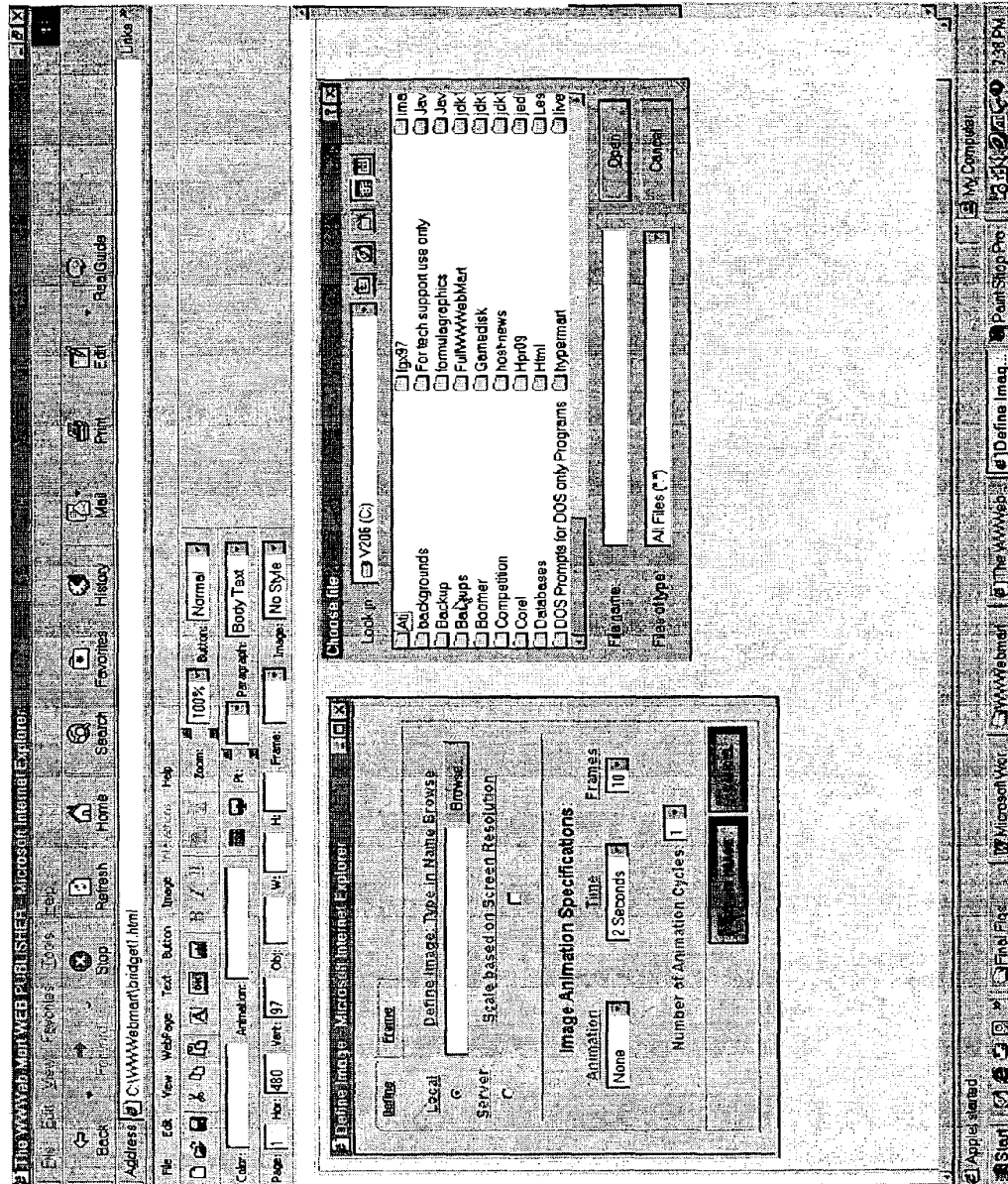


Fig. 22

## DYNAMIC WEB PAGE RESIZING PROCESS



**Fig. 23**

U.S. Patent

Sep. 22, 2009

Sheet 29 of 68

US 7,594,168 B2

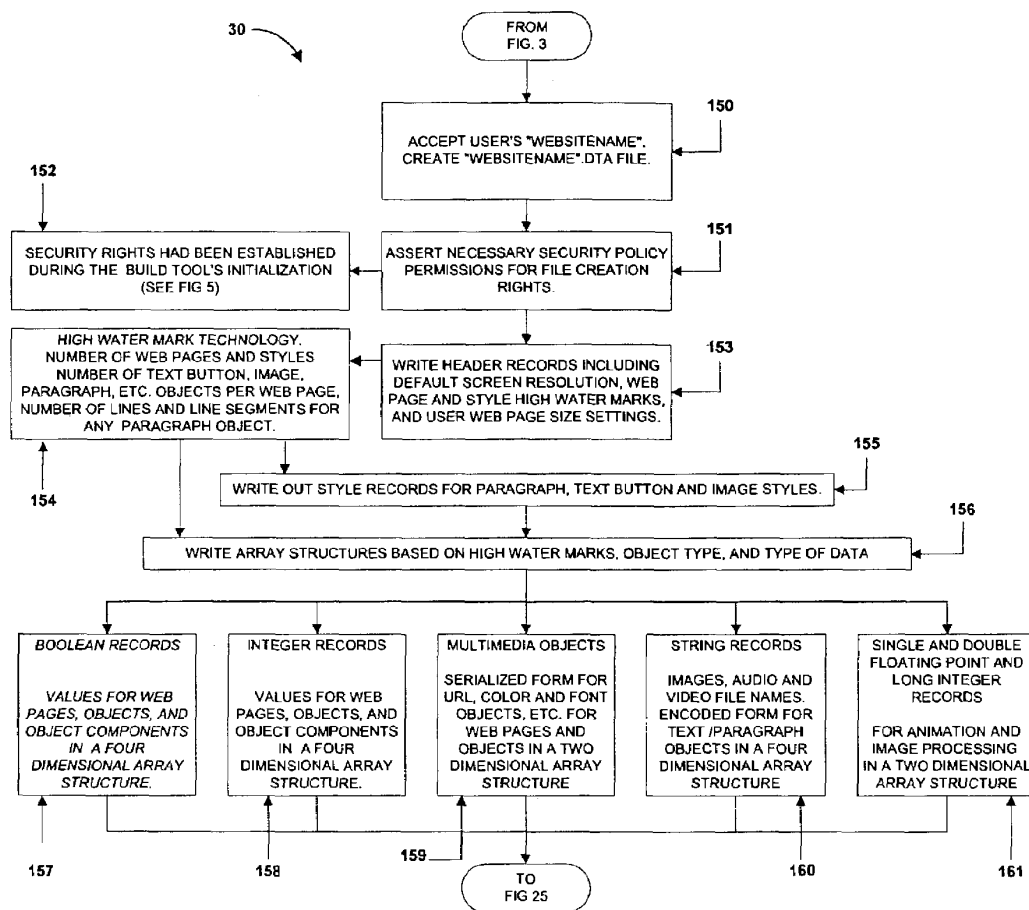


Fig. 24

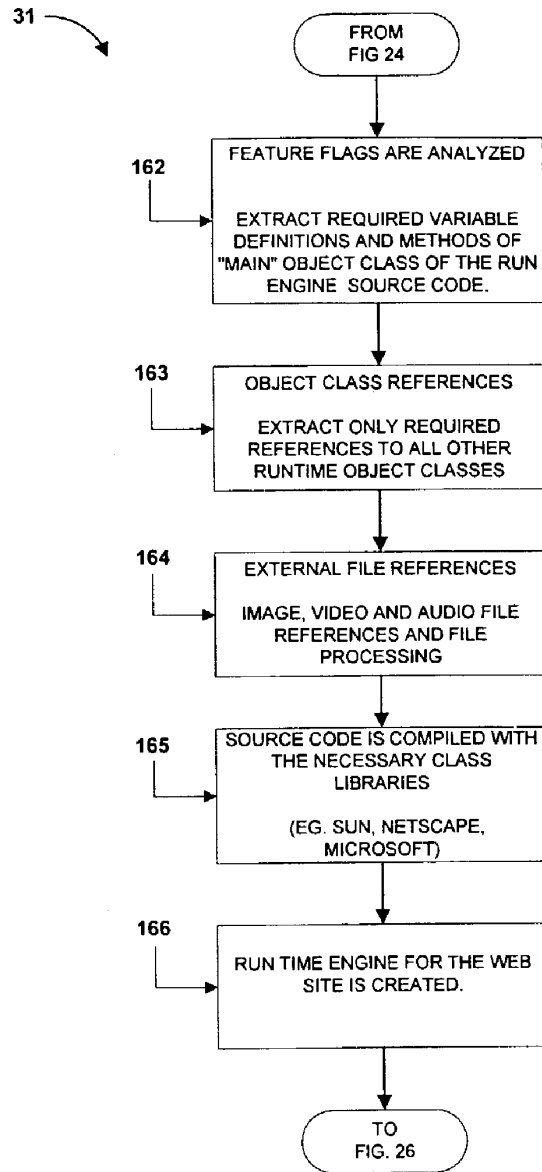
## EXTERNAL DATA BASE CREATION: SECURITY AND OPTIMIZATION TECHNIQUES

U.S. Patent

Sep. 22, 2009

Sheet 30 of 68

US 7,594,168 B2



*Fig. 25*

**CREATE CUSTOMIZED AND OPTIMIZED  
RUNTIME ENGINE**

U.S. Patent

Sep. 22, 2009

Sheet 31 of 68

US 7,594,168 B2

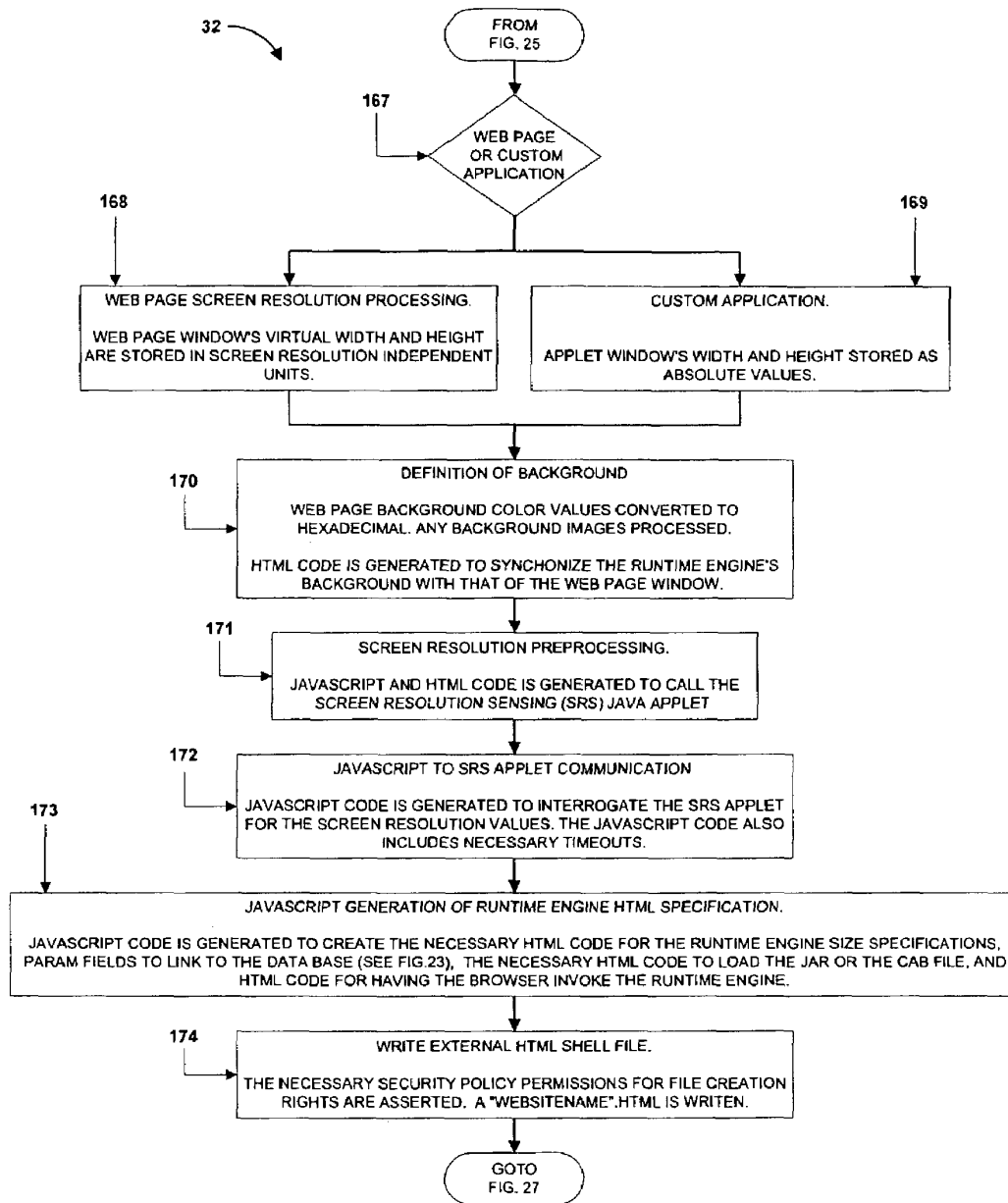


Fig. 26

## CREATE THE HTML SHELL FILE

U.S. Patent

Sep. 22, 2009

Sheet 32 of 68

US 7,594,168 B2

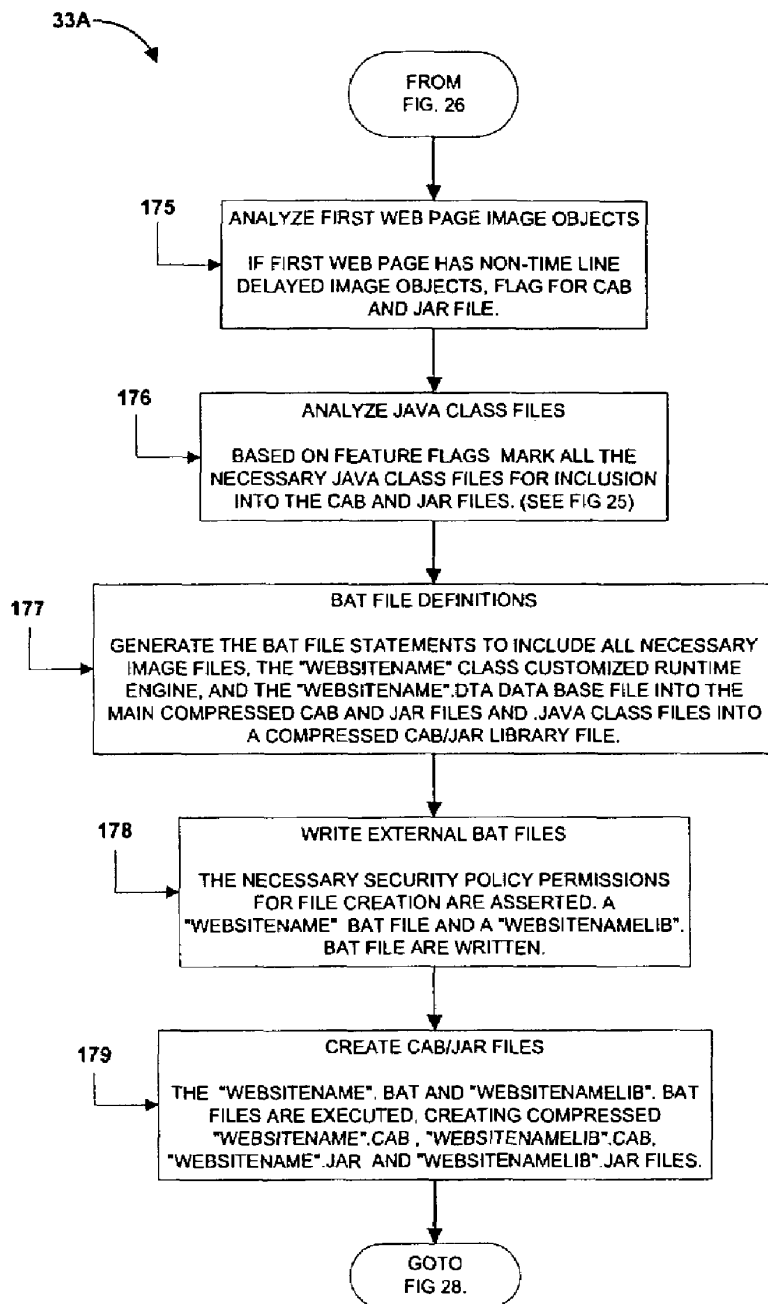


Fig. 27

**CREATE THE CAB/JAR FILES**

U.S. Patent

Sep. 22, 2009

Sheet 33 of 68

US 7,594,168 B2

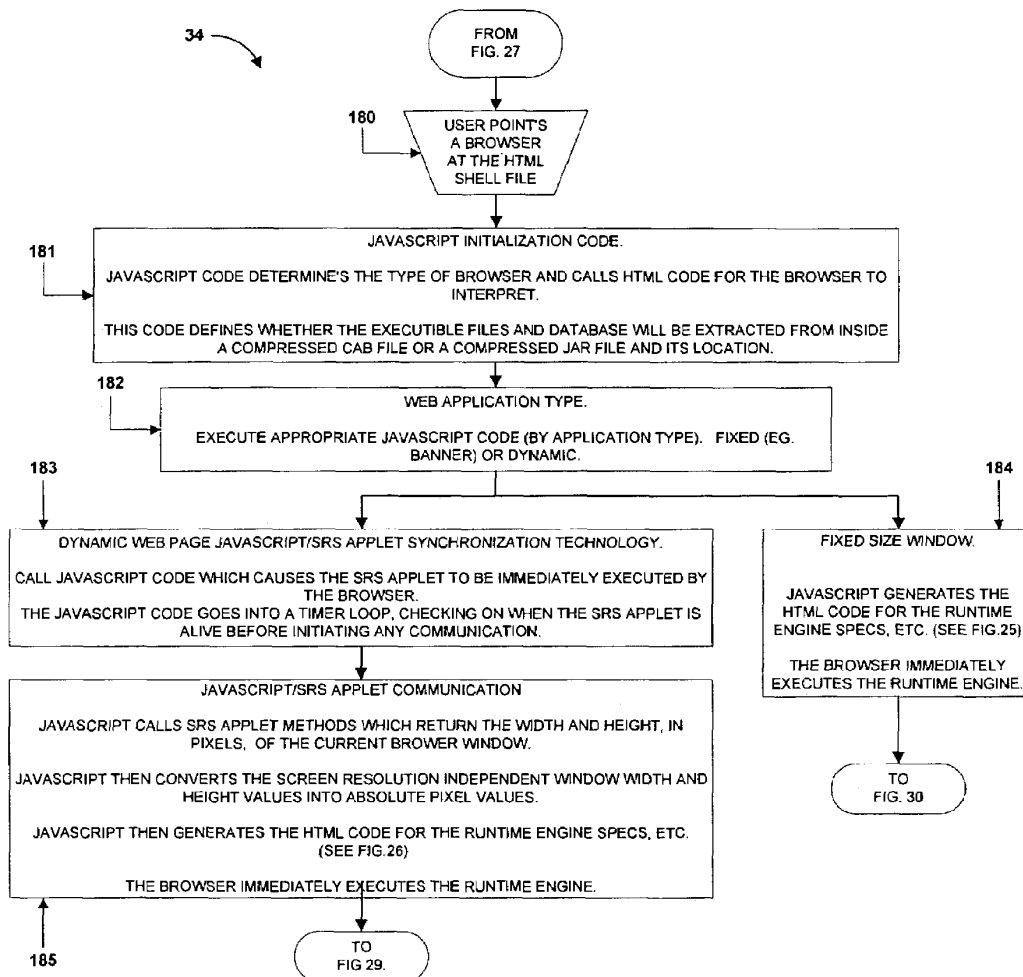


Fig. 28

## WEB PAGE SIZE GENERATION TECHNOLOGY

U.S. Patent

Sep. 22, 2009

Sheet 34 of 68

US 7,594,168 B2

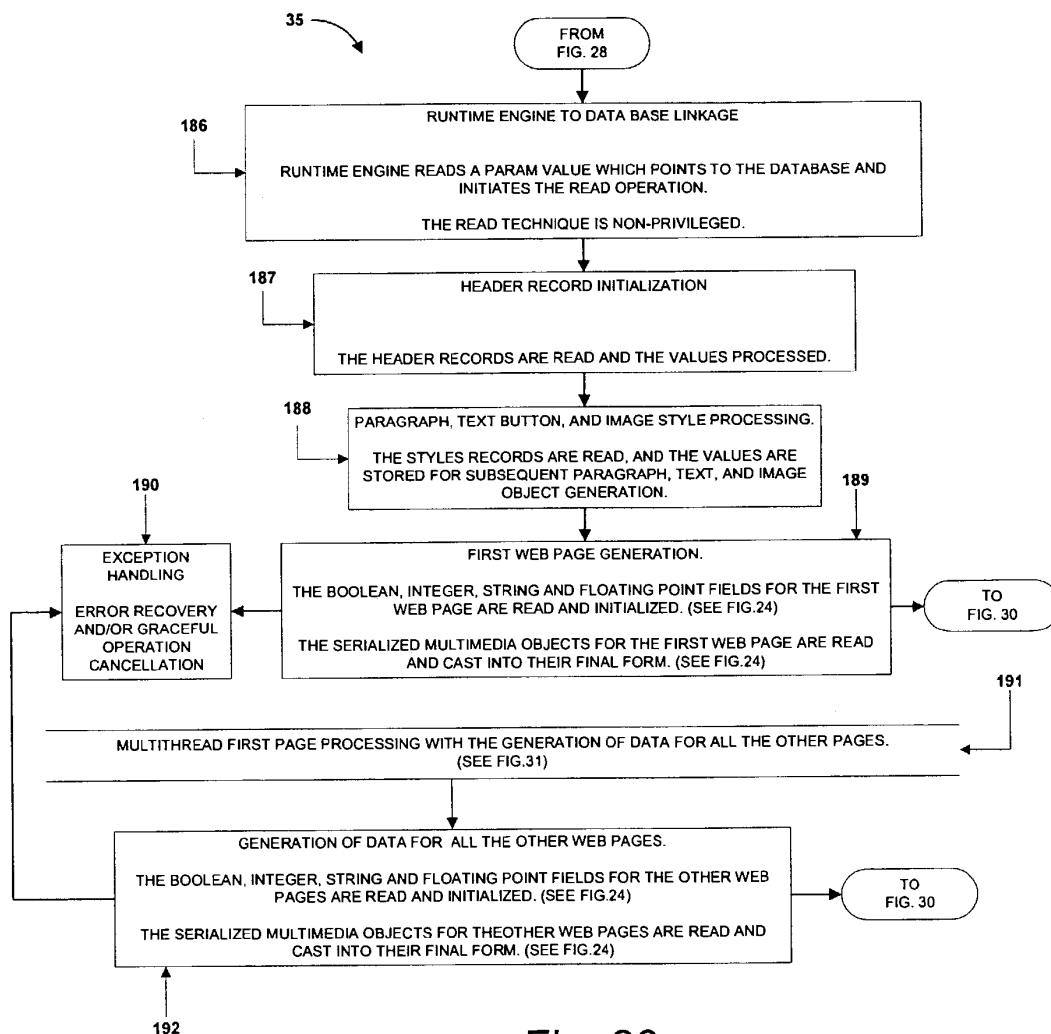


Fig. 29

**READ DATA BASE AND GENERATE  
NECESSARY OBJECTS.**



U.S. Patent

Sep. 22, 2009

Sheet 35 of 68

US 7,594,168 B2

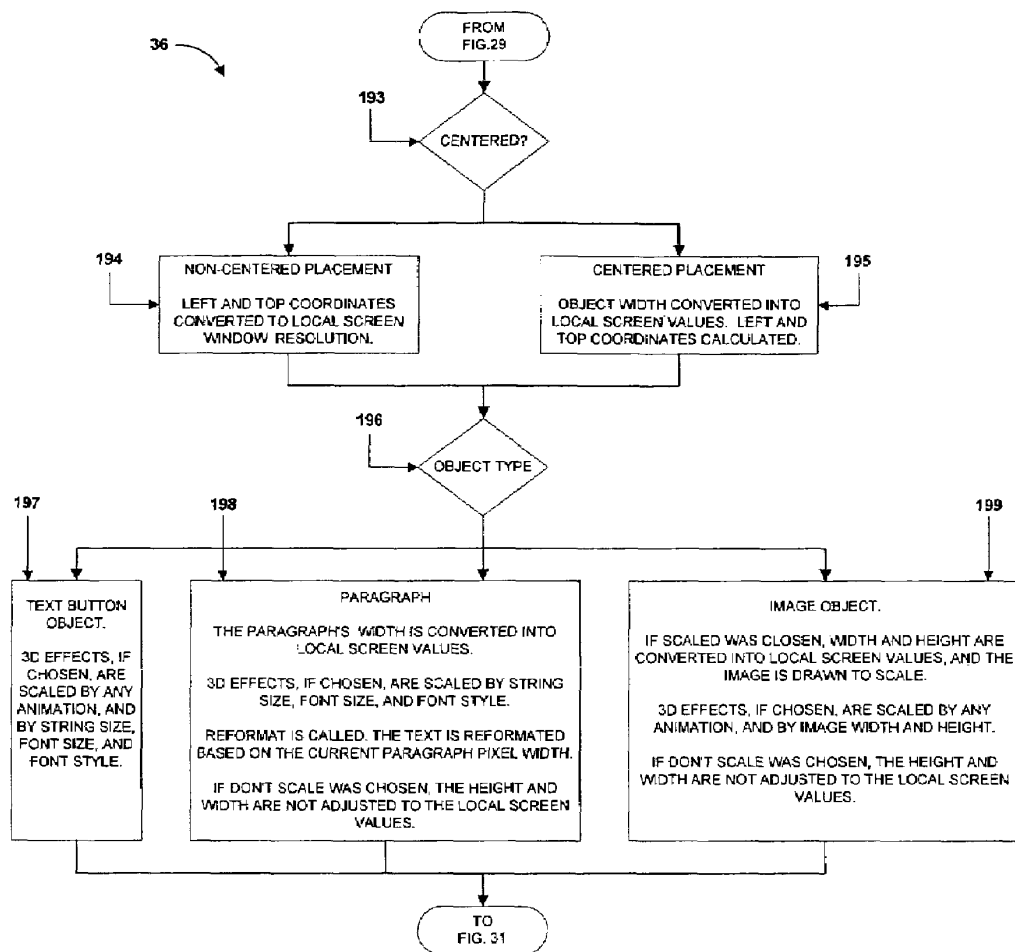


Fig. 30

## WEB PAGE GENERATION WITH SCALING TECHNOLOGY

U.S. Patent

Sep. 22, 2009

Sheet 36 of 68

US 7,594,168 B2

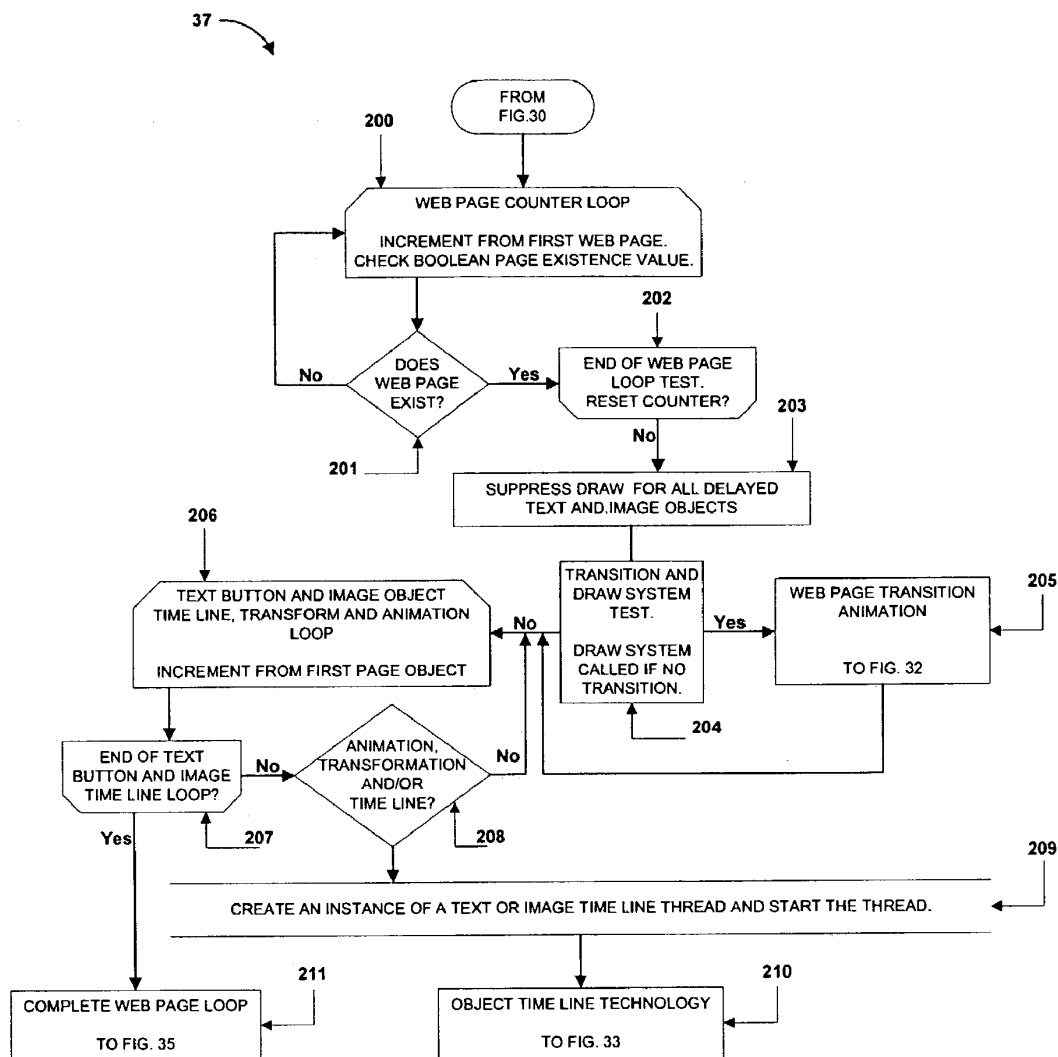


Fig. 31

## THE MULTILEVEL WEB PAGE AND OBJECT THREAD TECHNOLOGY

U.S. Patent

Sep. 22, 2009

Sheet 37 of 68

US 7,594,168 B2

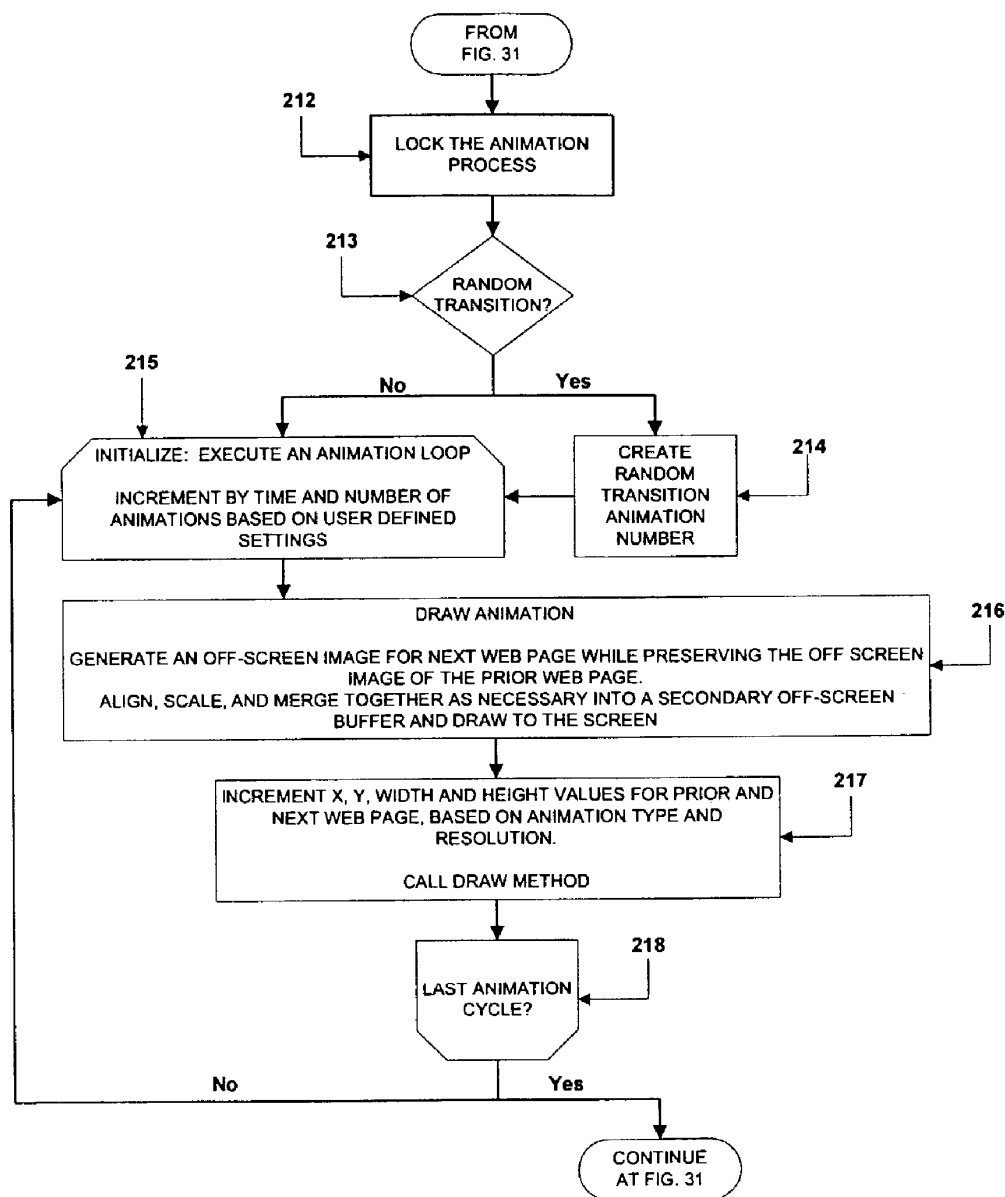


Fig. 32

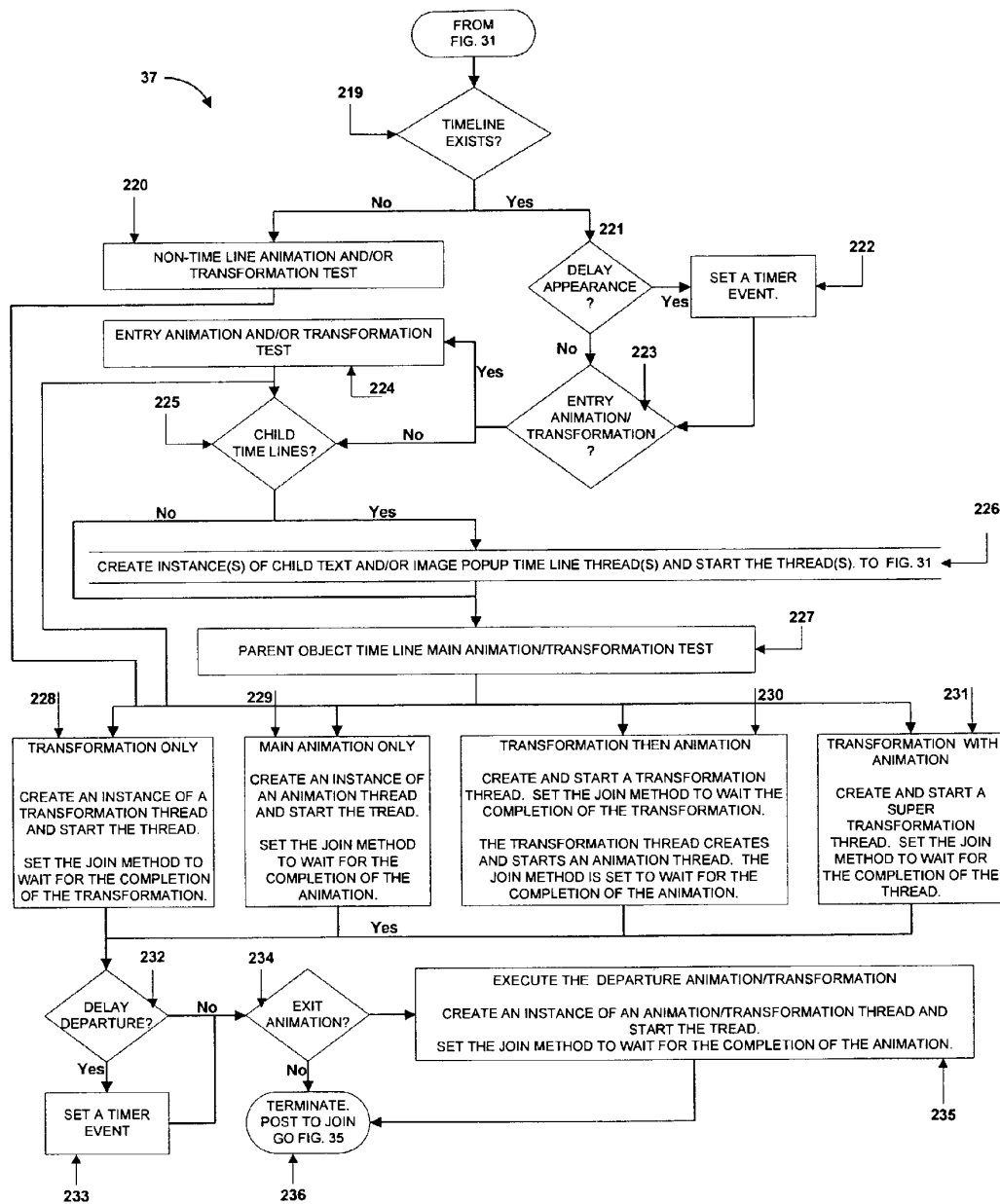
## WEB PAGE TRANSITION ANIMATION

U.S. Patent

Sep. 22, 2009

Sheet 38 of 68

US 7,594,168 B2

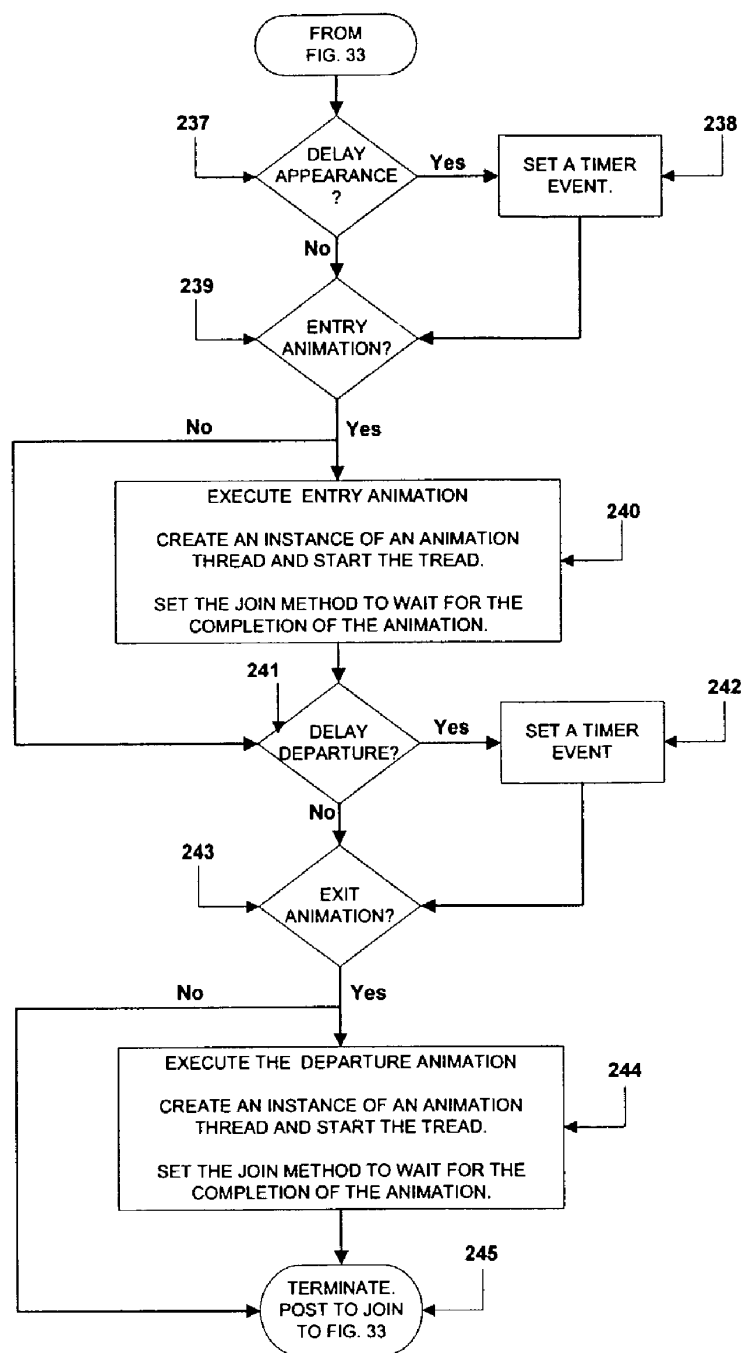


U.S. Patent

Sep. 22, 2009

Sheet 39 of 68

US 7,594,168 B2

*Fig. 34*

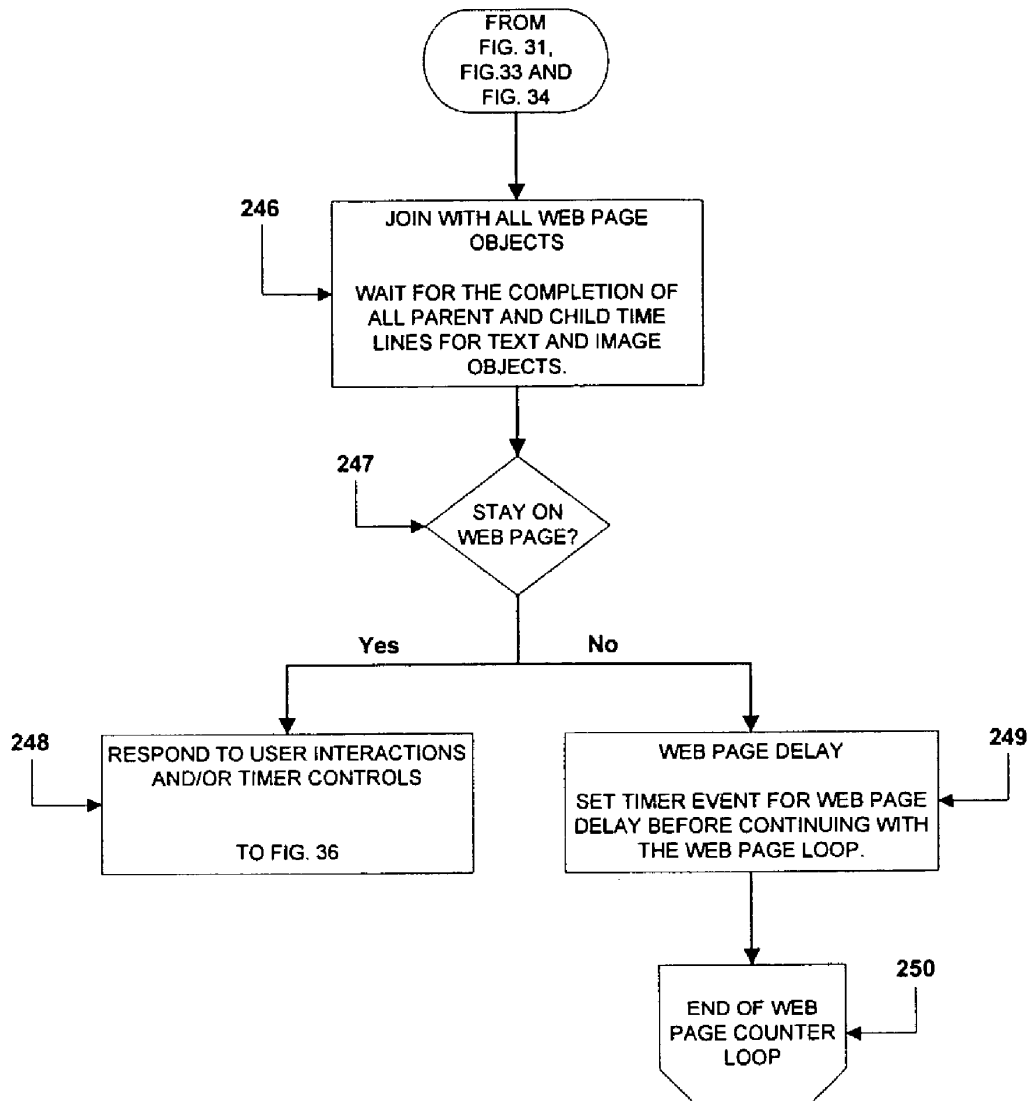
**CHILD TIME LINES FOR TEXT  
BUTTON AND IMAGE OBJECTS**

U.S. Patent

Sep. 22, 2009

Sheet 40 of 68

US 7,594,168 B2



*Fig. 35*

## COMPLETE WEB PAGE AND OBJECT THREAD LOOP

U.S. Patent

Sep. 22, 2009

Sheet 41 of 68

US 7,594,168 B2

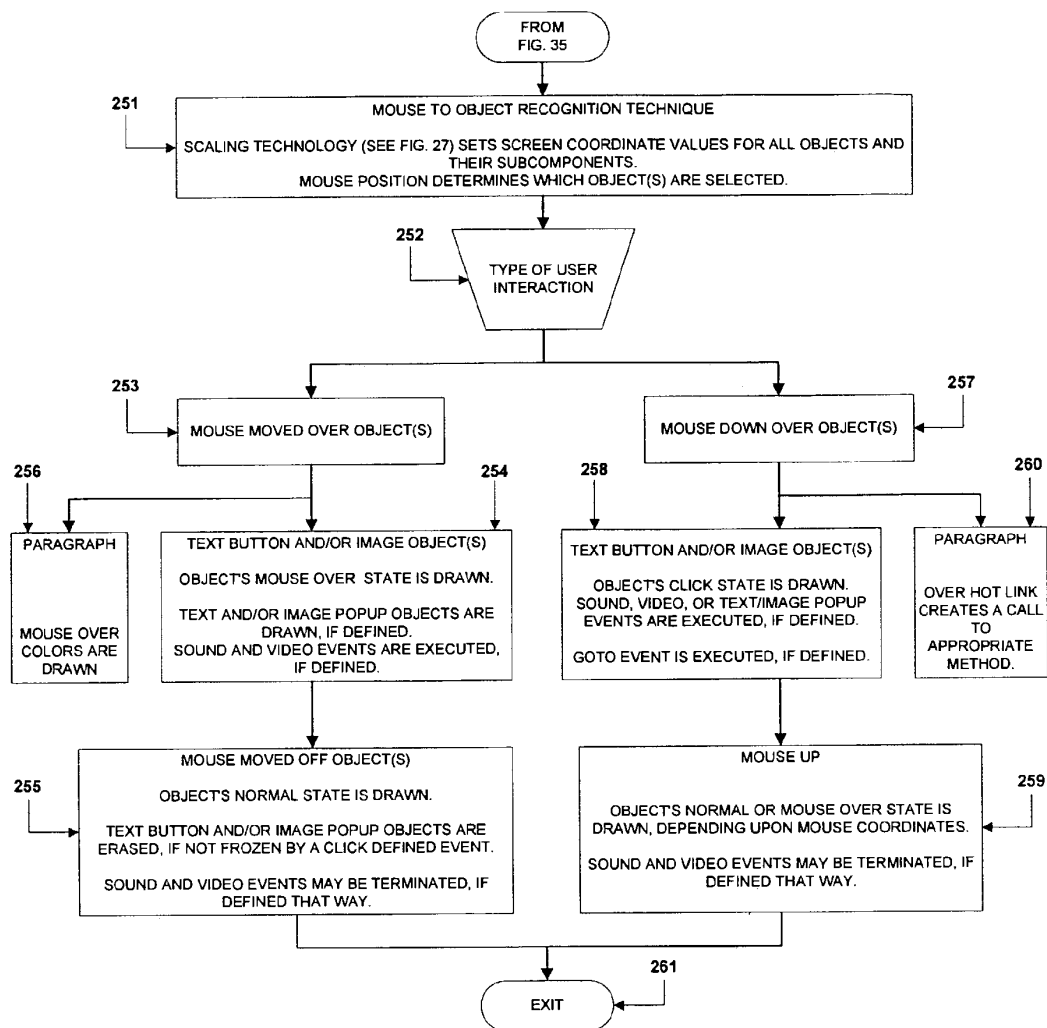


Fig. 36

**RESPOND TO USER INTERACTIONS**

U.S. Patent

Sep. 22, 2009

Sheet 42 of 68

US 7,594,168 B2

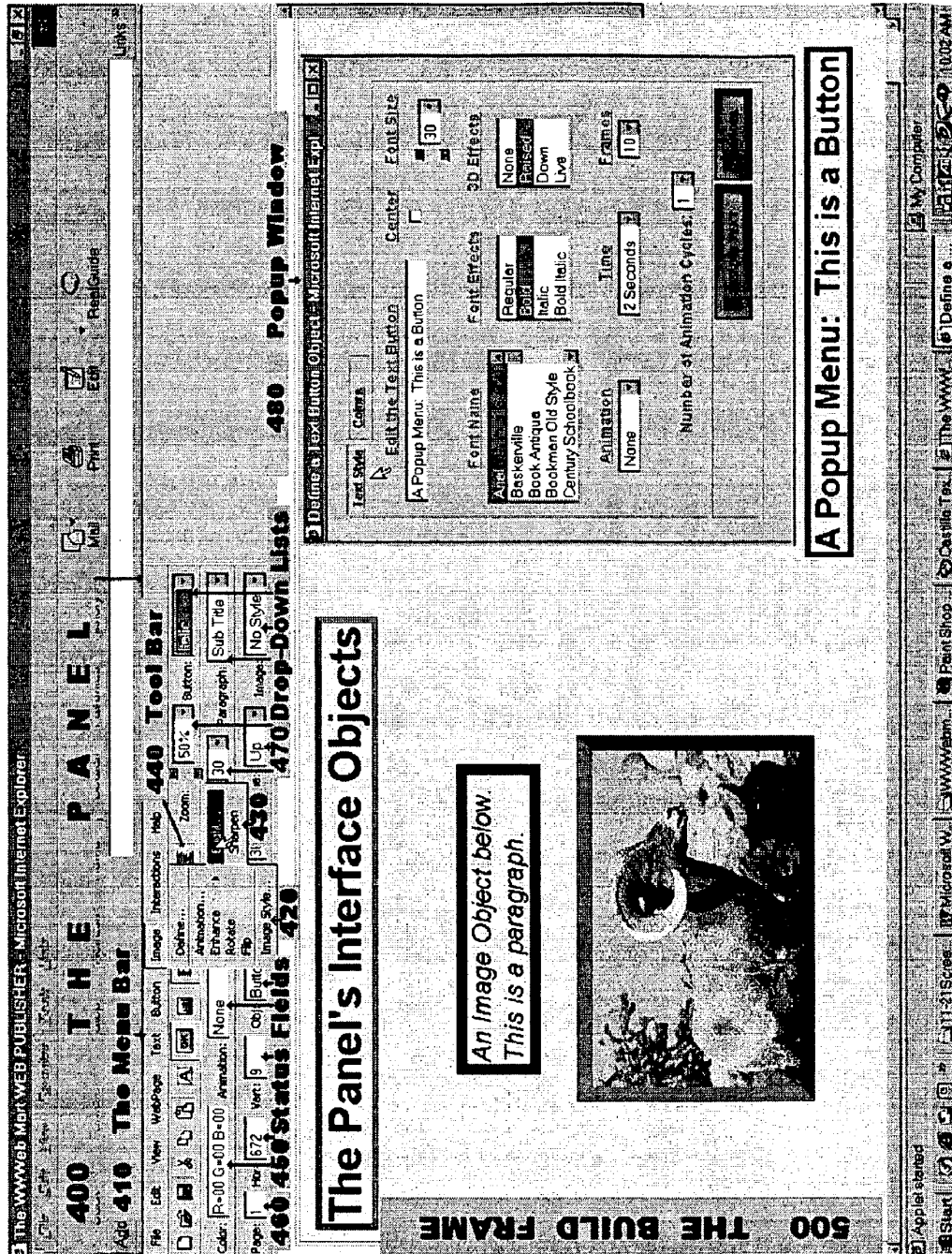


Fig. 37



U.S. Patent

Sep. 22, 2009

Sheet 43 of 68

US 7,594,168 B2

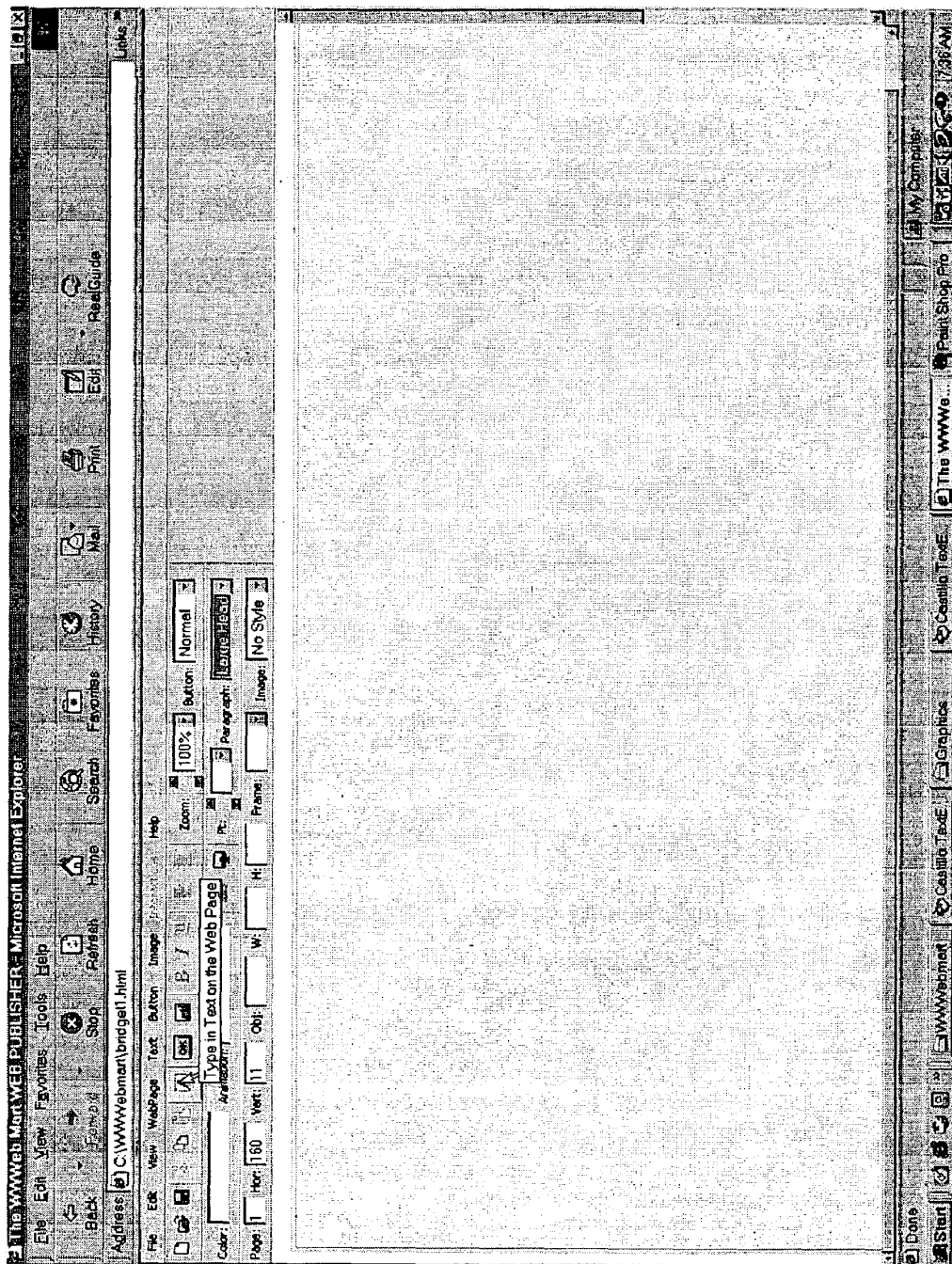


Fig. 38

U.S. Patent

Sep. 22, 2009

Sheet 44 of 68

US 7,594,168 B2

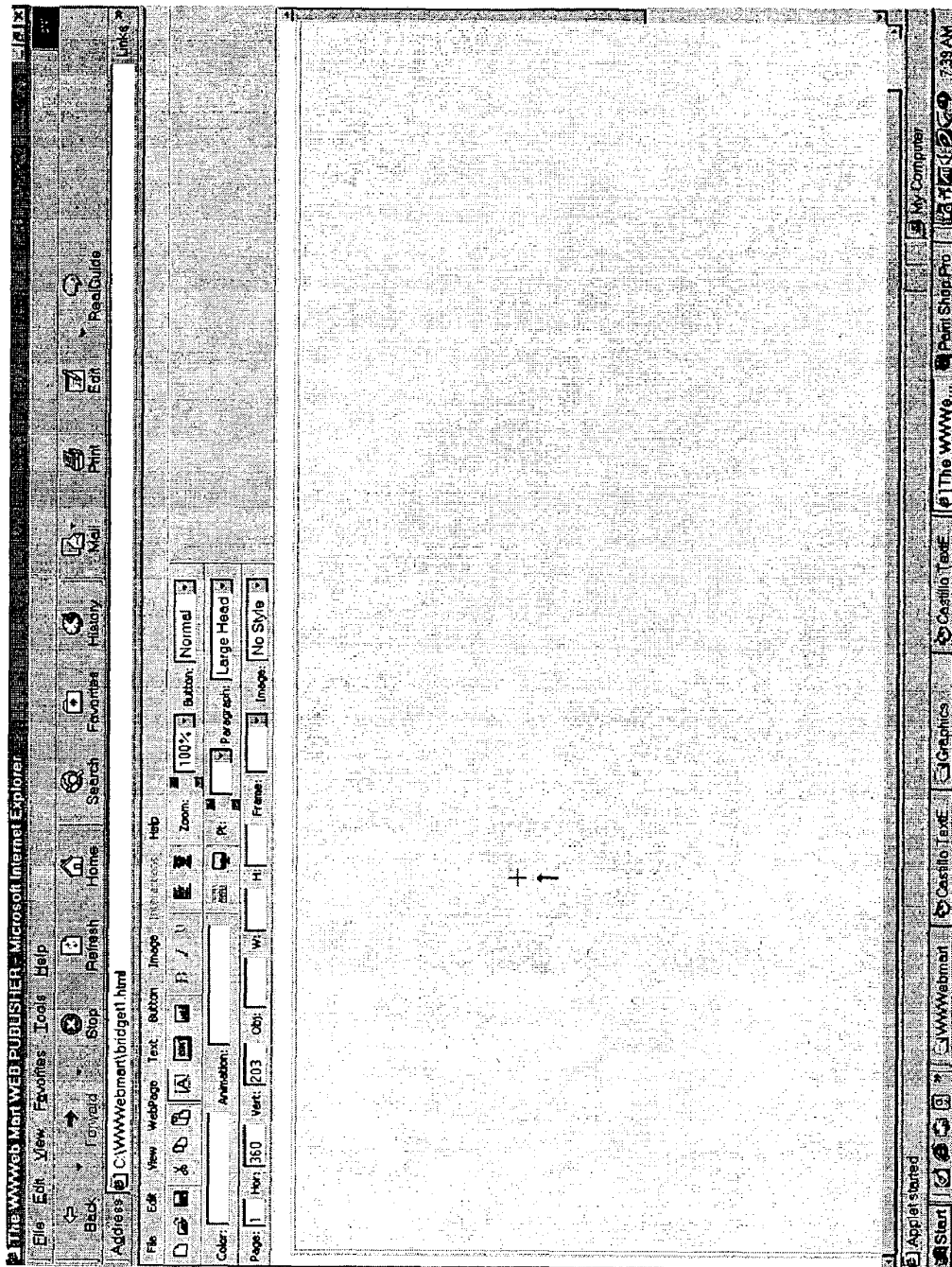


Fig. 39

U.S. Patent

Sep. 22, 2009

Sheet 45 of 68

US 7,594,168 B2

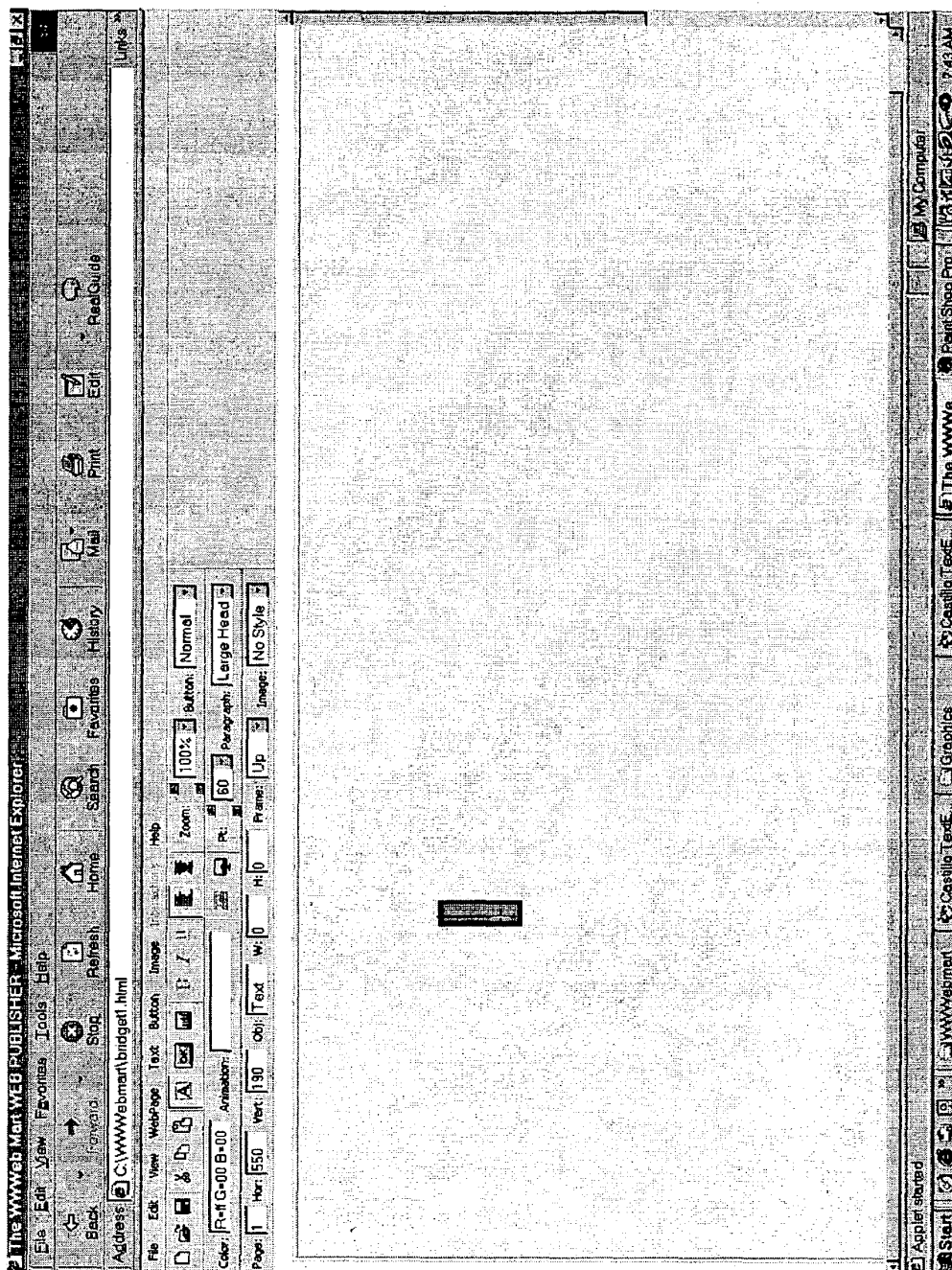


Fig. 40

U.S. Patent

Sep. 22, 2009

Sheet 46 of 68

US 7,594,168 B2

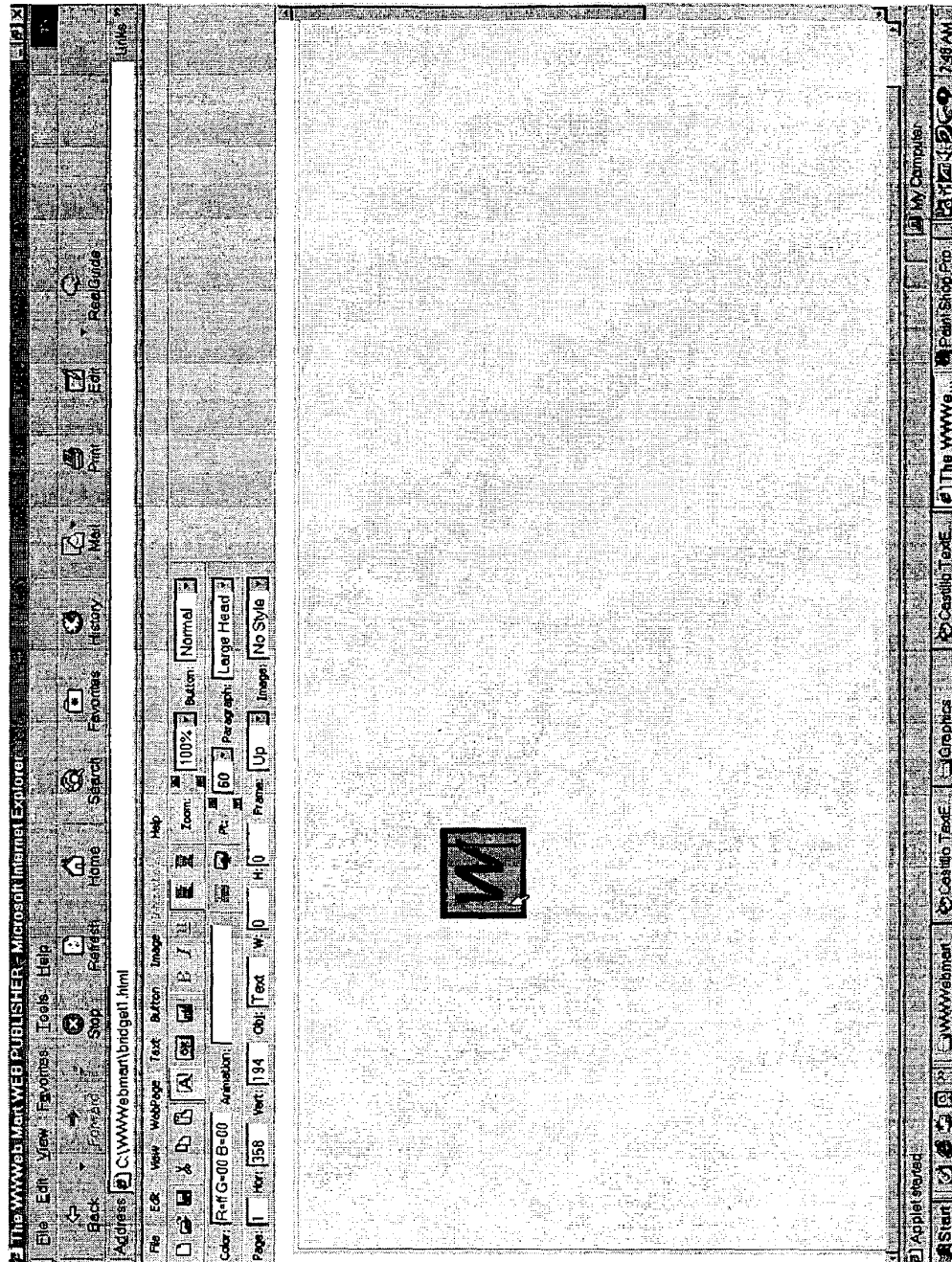


Fig. 41

U.S. Patent

Sep. 22, 2009

Sheet 47 of 68

US 7,594,168 B2

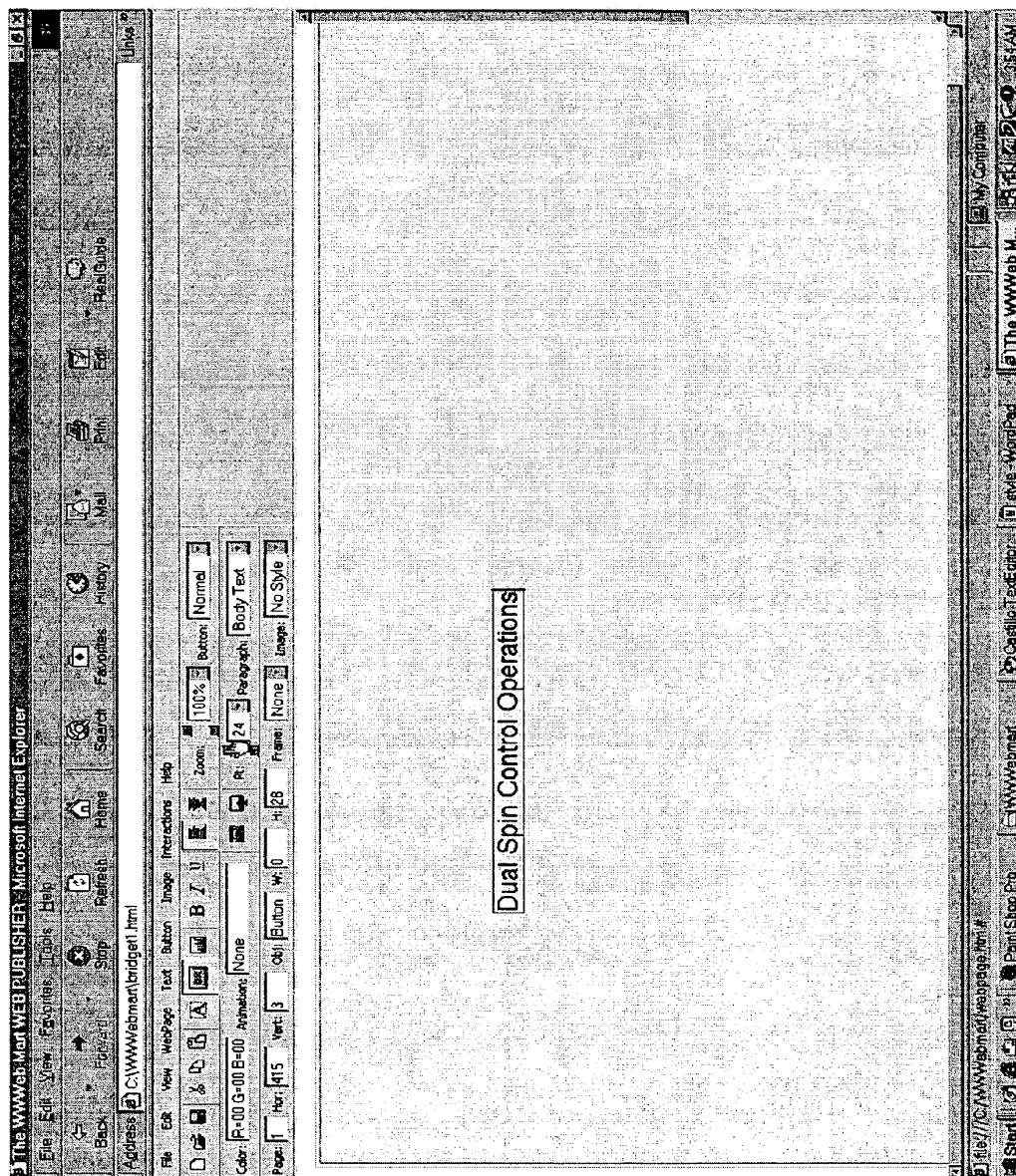


Fig. 42



U.S. Patent

Sep. 22, 2009

Sheet 48 of 68

US 7,594,168 B2

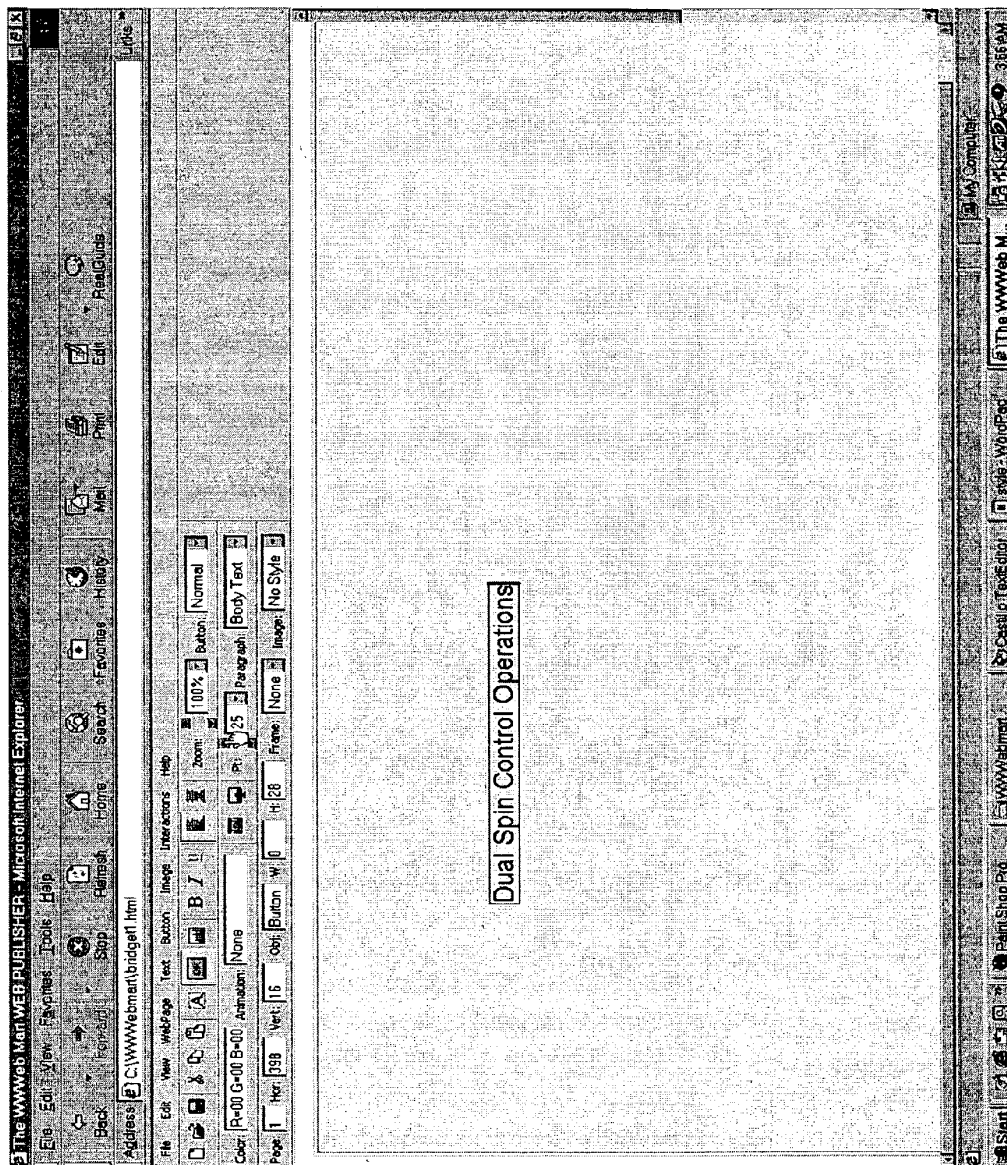


Fig. 43

U.S. Patent

Sep. 22, 2009

Sheet 49 of 68

US 7,594,168 B2

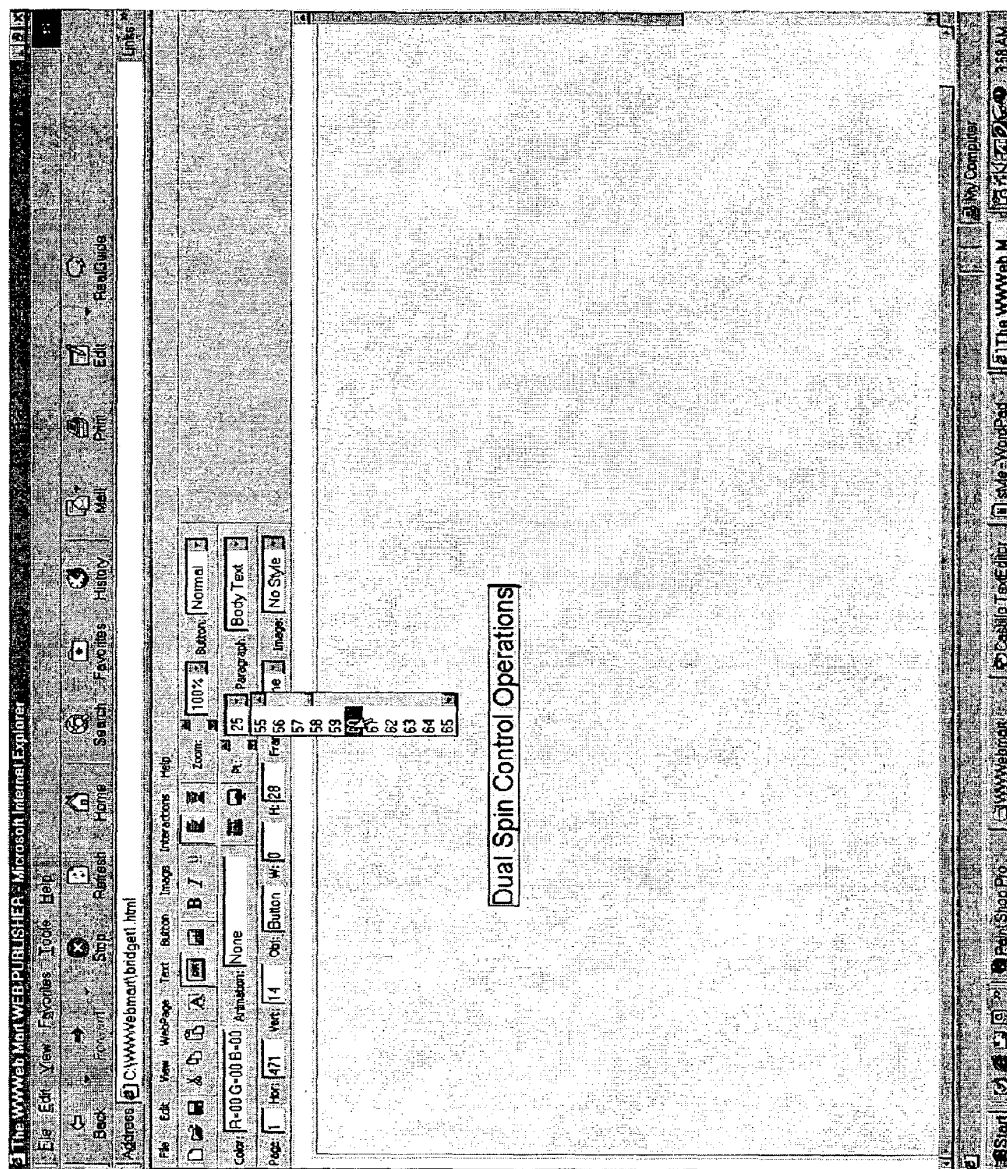


Fig. 44

U.S. Patent

Sep. 22, 2009

Sheet 50 of 68

US 7,594,168 B2

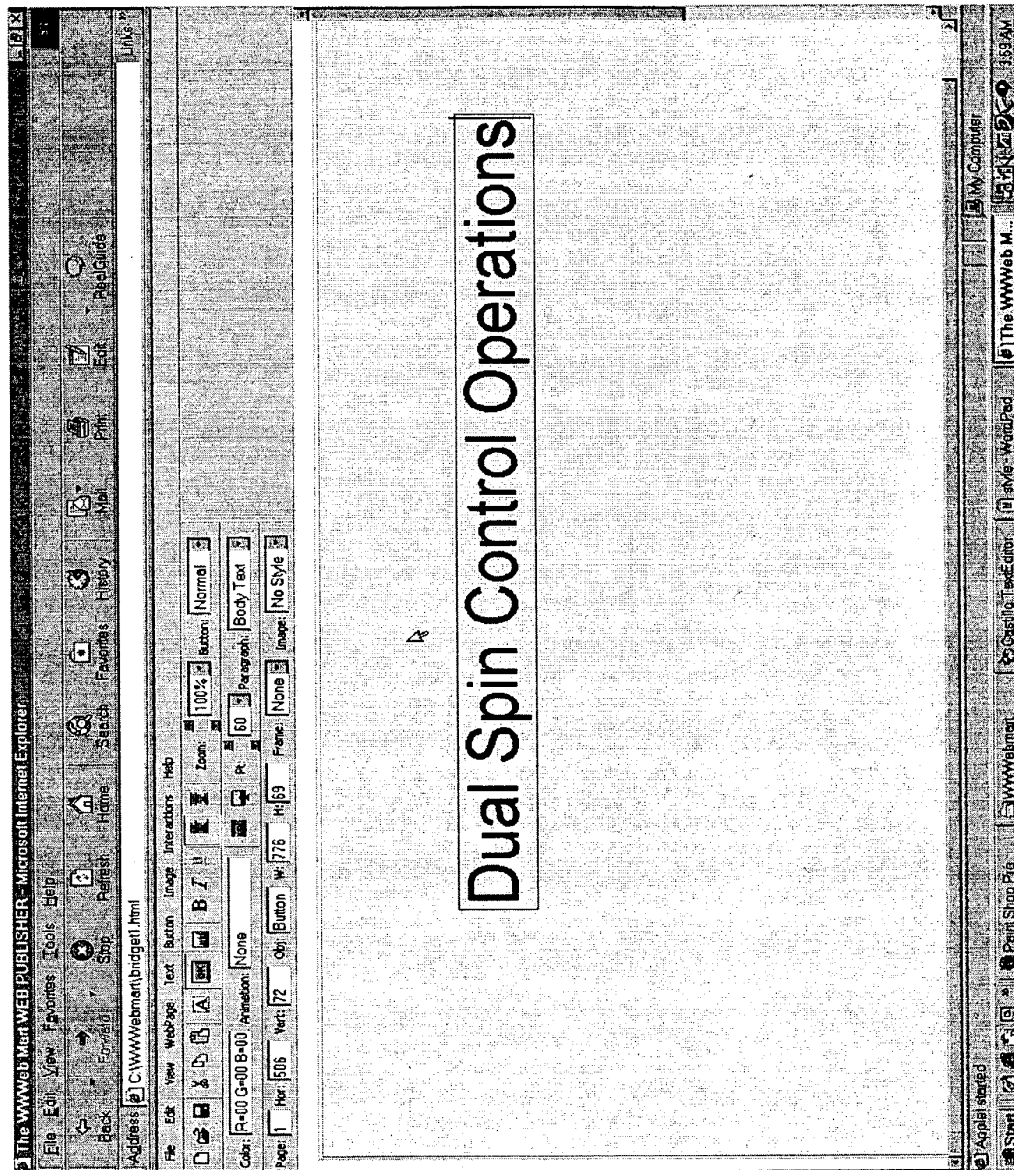


Fig. 45



U.S. Patent

Sep. 22, 2009

Sheet 51 of 68

US 7,594,168 B2

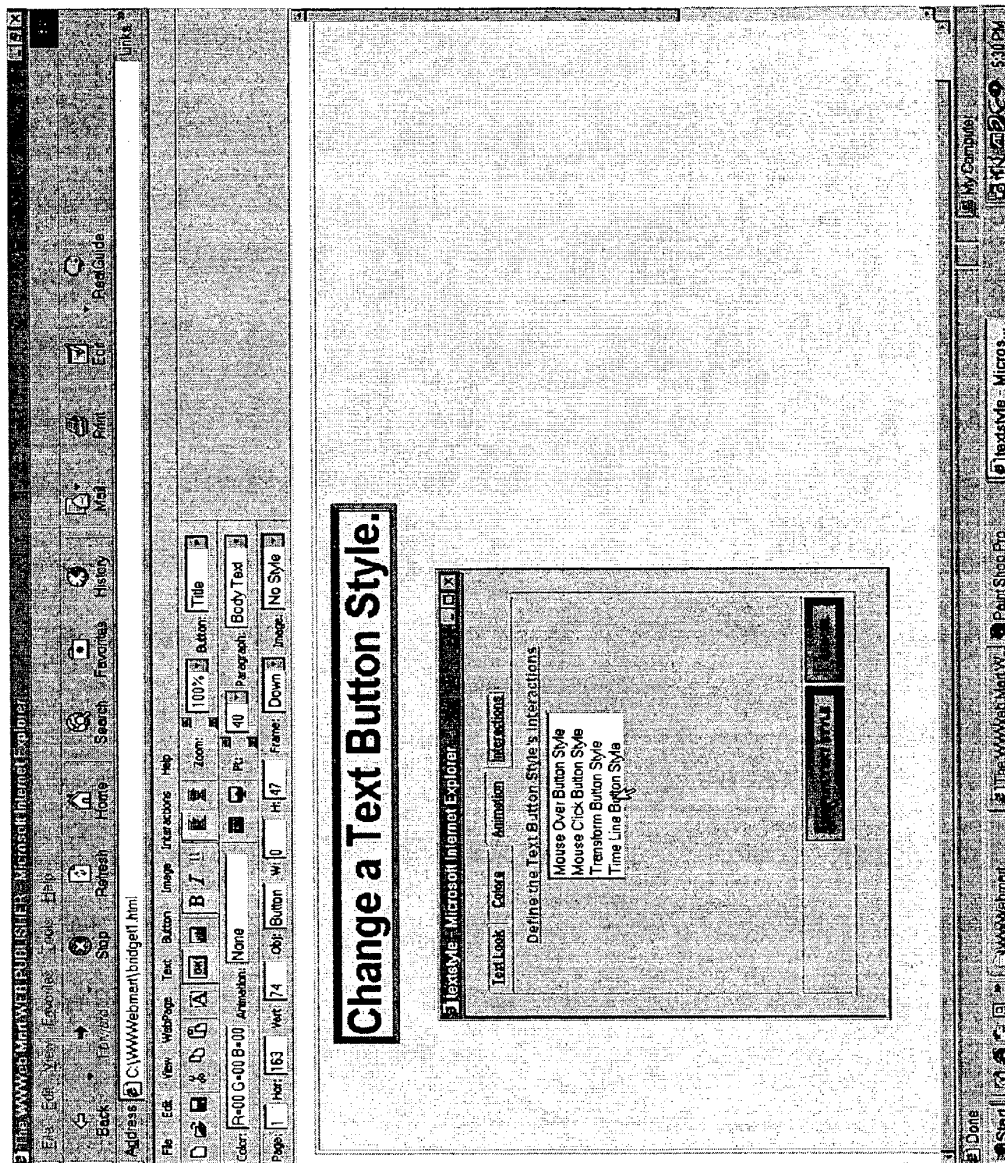


Fig. 46

U.S. Patent

Sep. 22, 2009

Sheet 52 of 68

US 7,594,168 B2

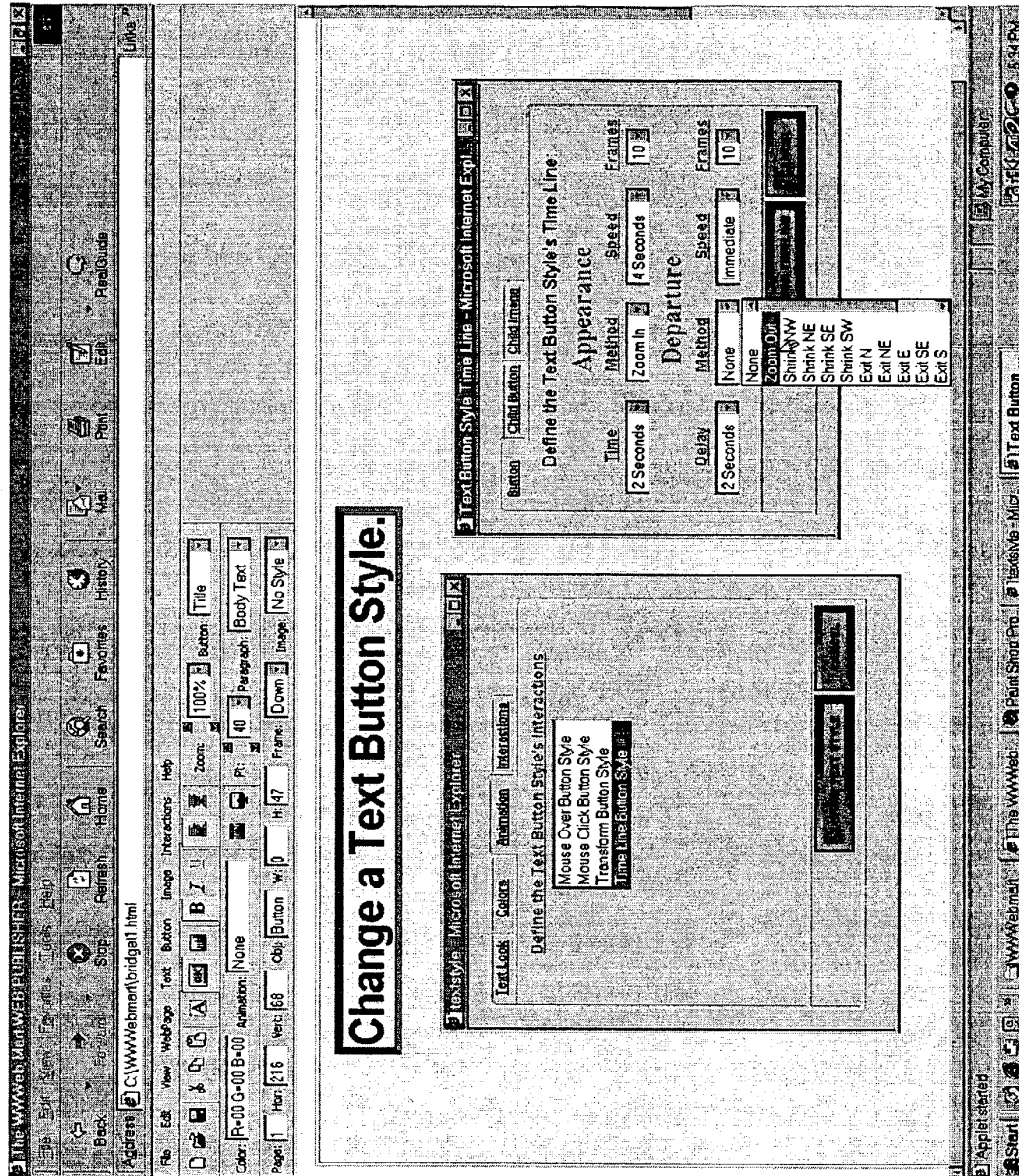


Fig. 47

U.S. Patent

Sep. 22, 2009

Sheet 53 of 68

US 7,594,168 B2

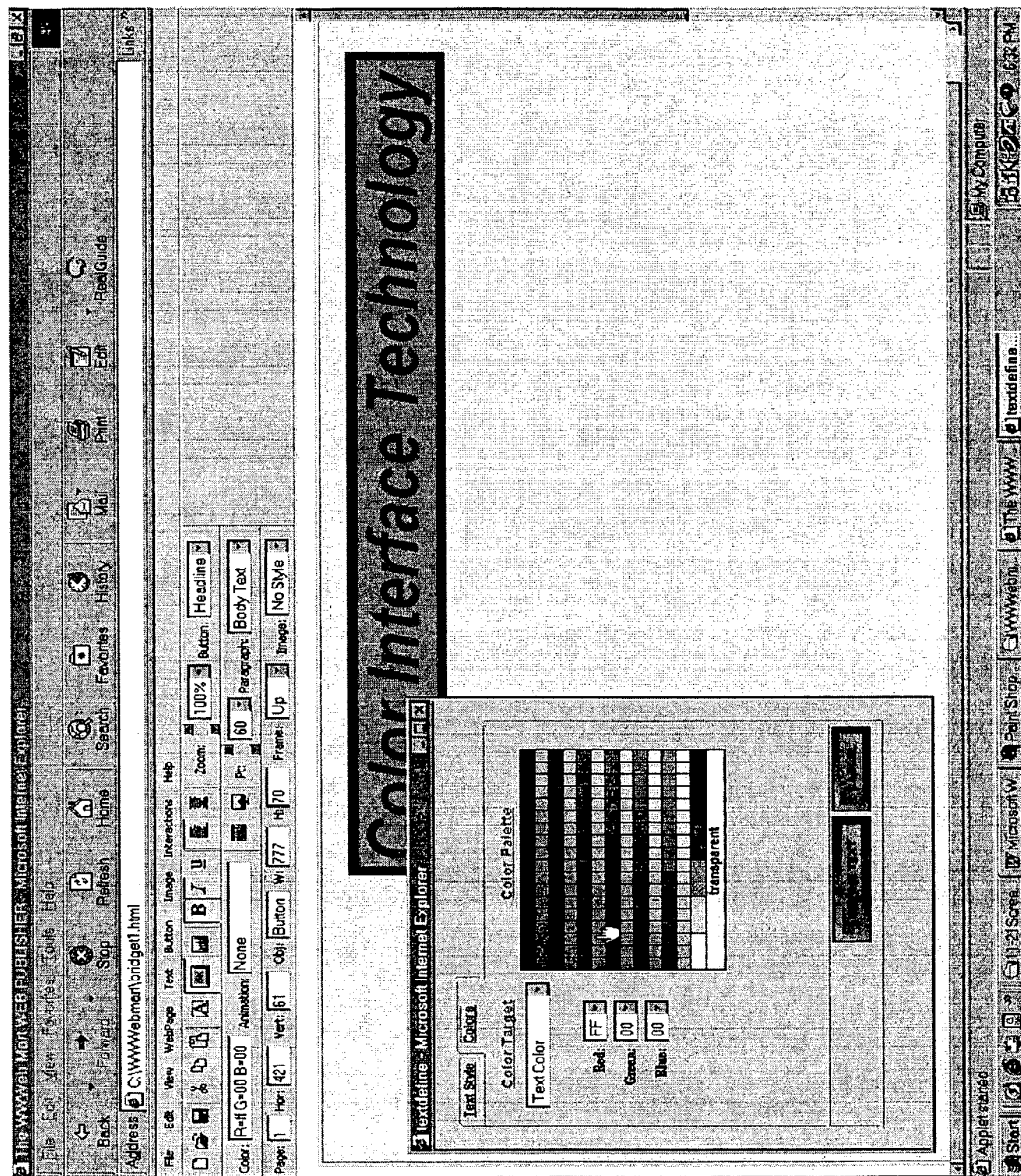


Fig. 48

U.S. Patent

Sep. 22, 2009

Sheet 54 of 68

US 7,594,168 B2

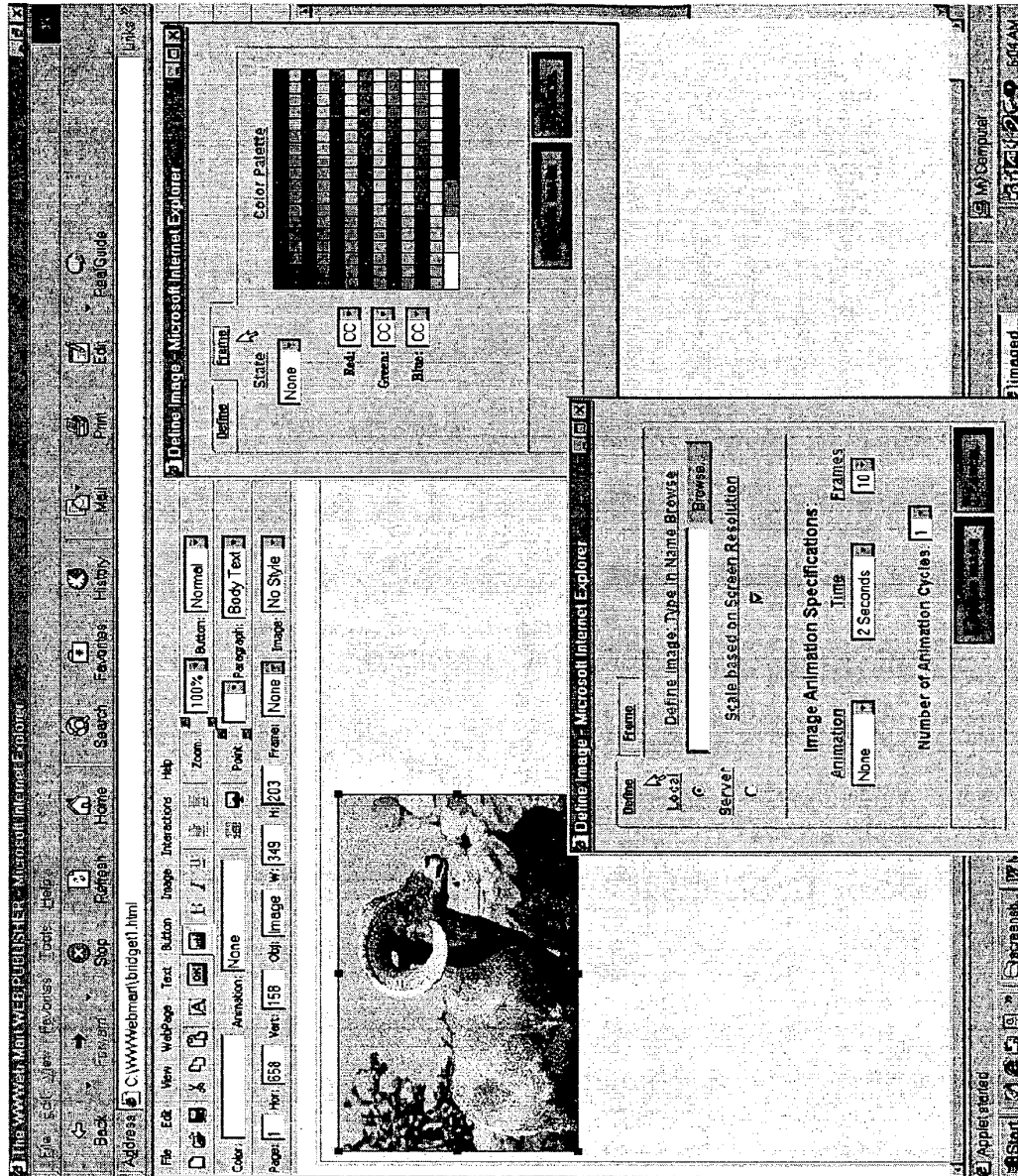


Fig. 49

U.S. Patent

Sep. 22, 2009

Sheet 55 of 68

US 7,594,168 B2

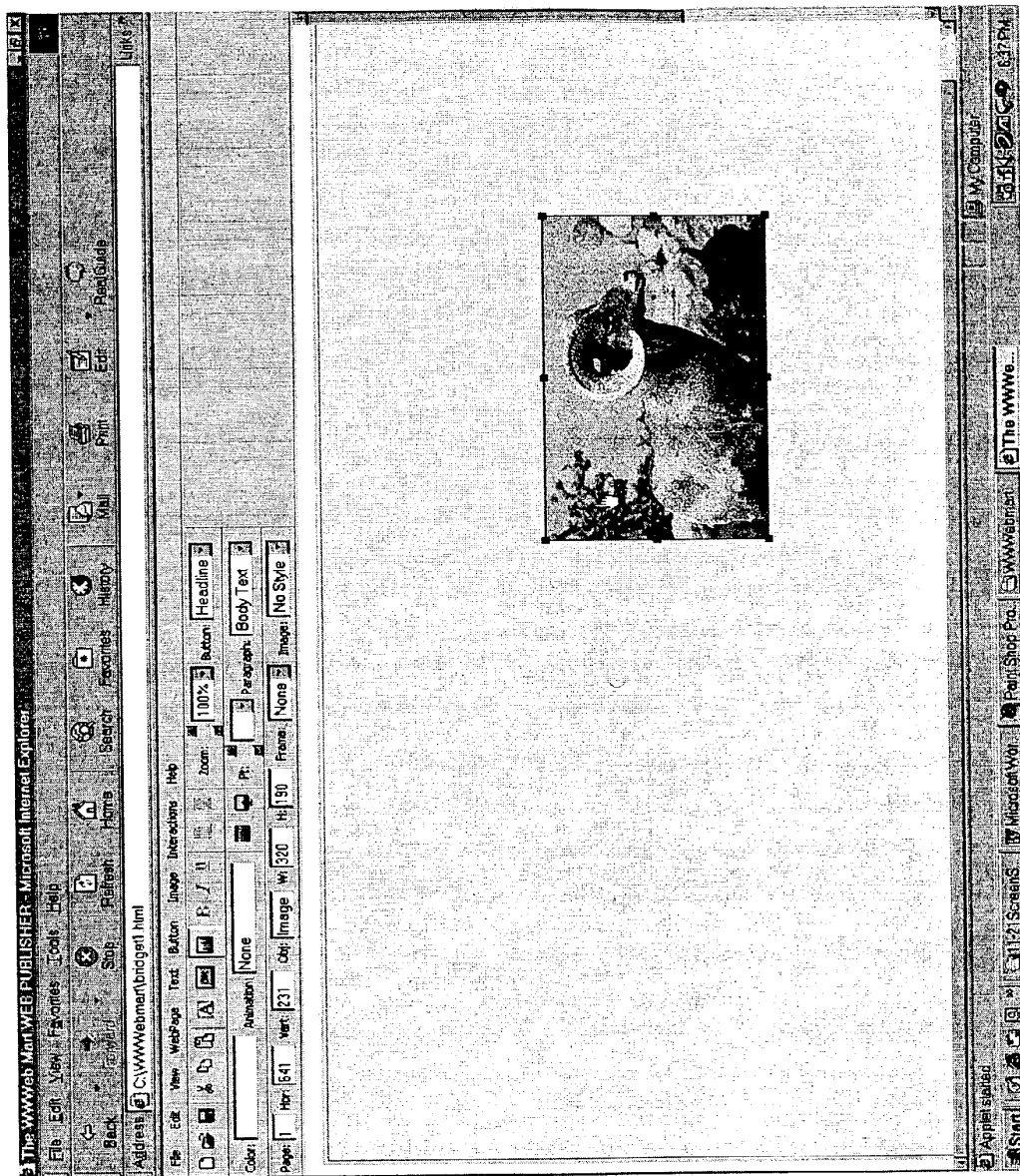


Fig. 50



U.S. Patent

Sep. 22, 2009

Sheet 56 of 68

US 7,594,168 B2

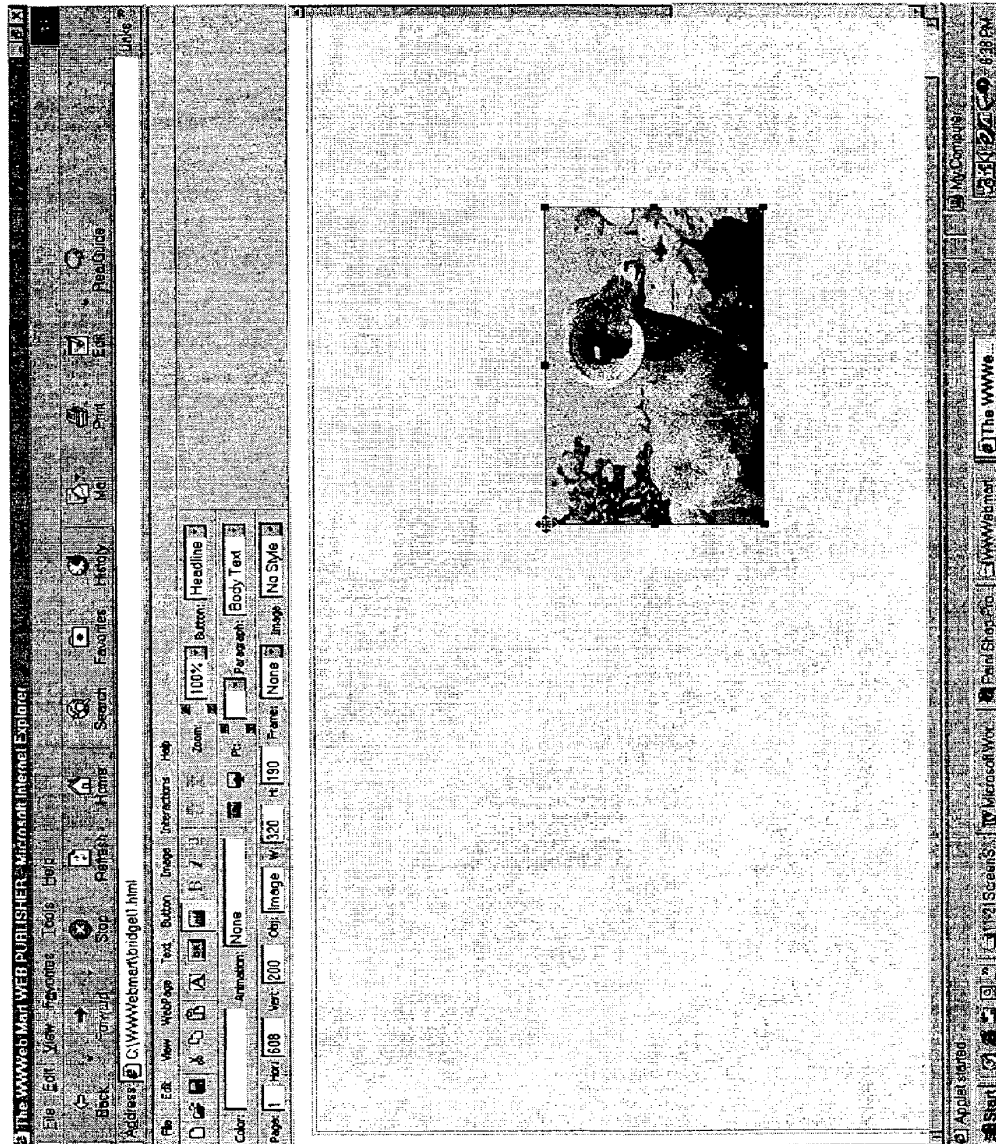


Fig. 51

U.S. Patent

Sep. 22, 2009

Sheet 57 of 68

US 7,594,168 B2

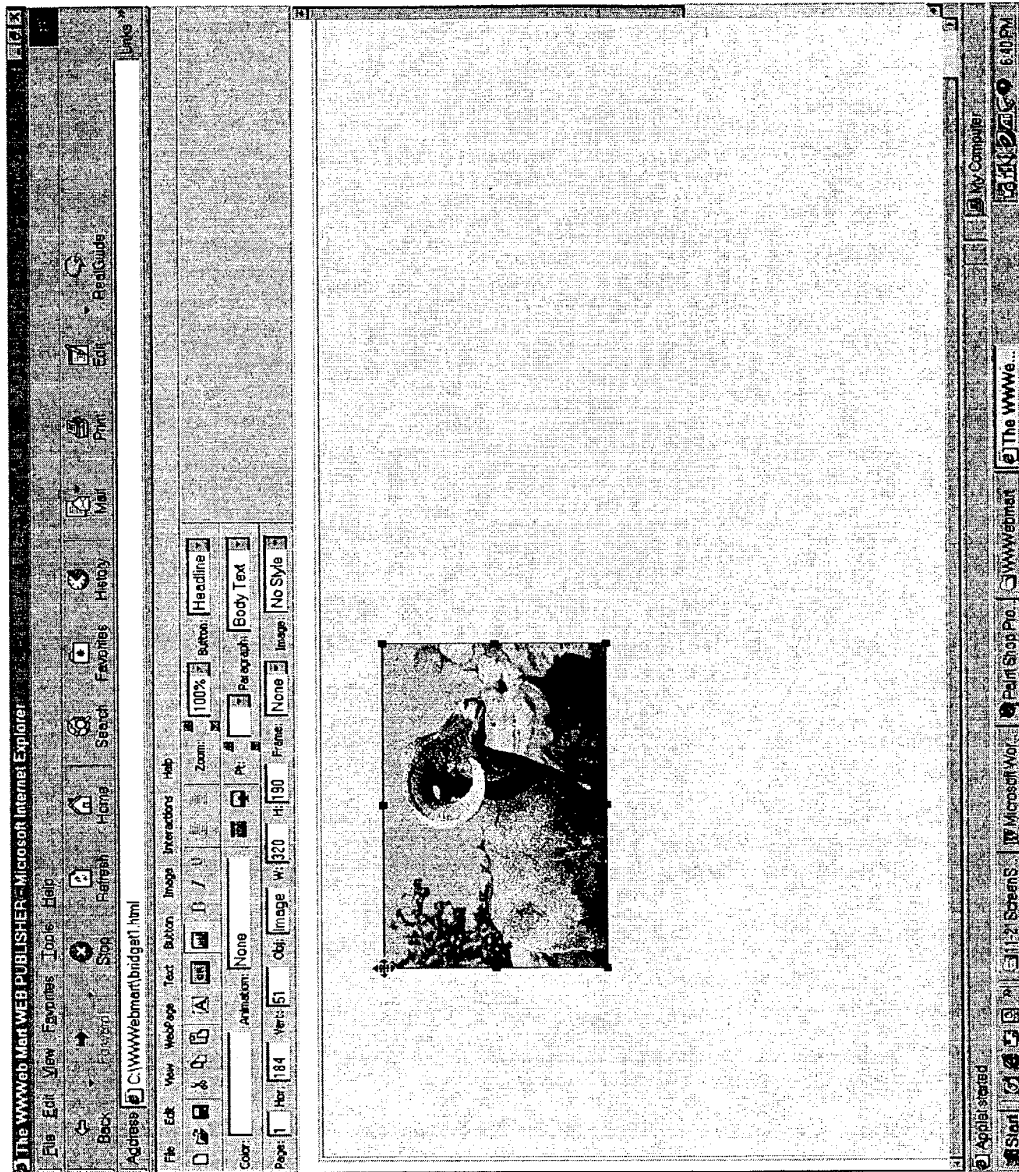


Fig. 52

U.S. Patent

Sep. 22, 2009

Sheet 58 of 68

US 7,594,168 B2

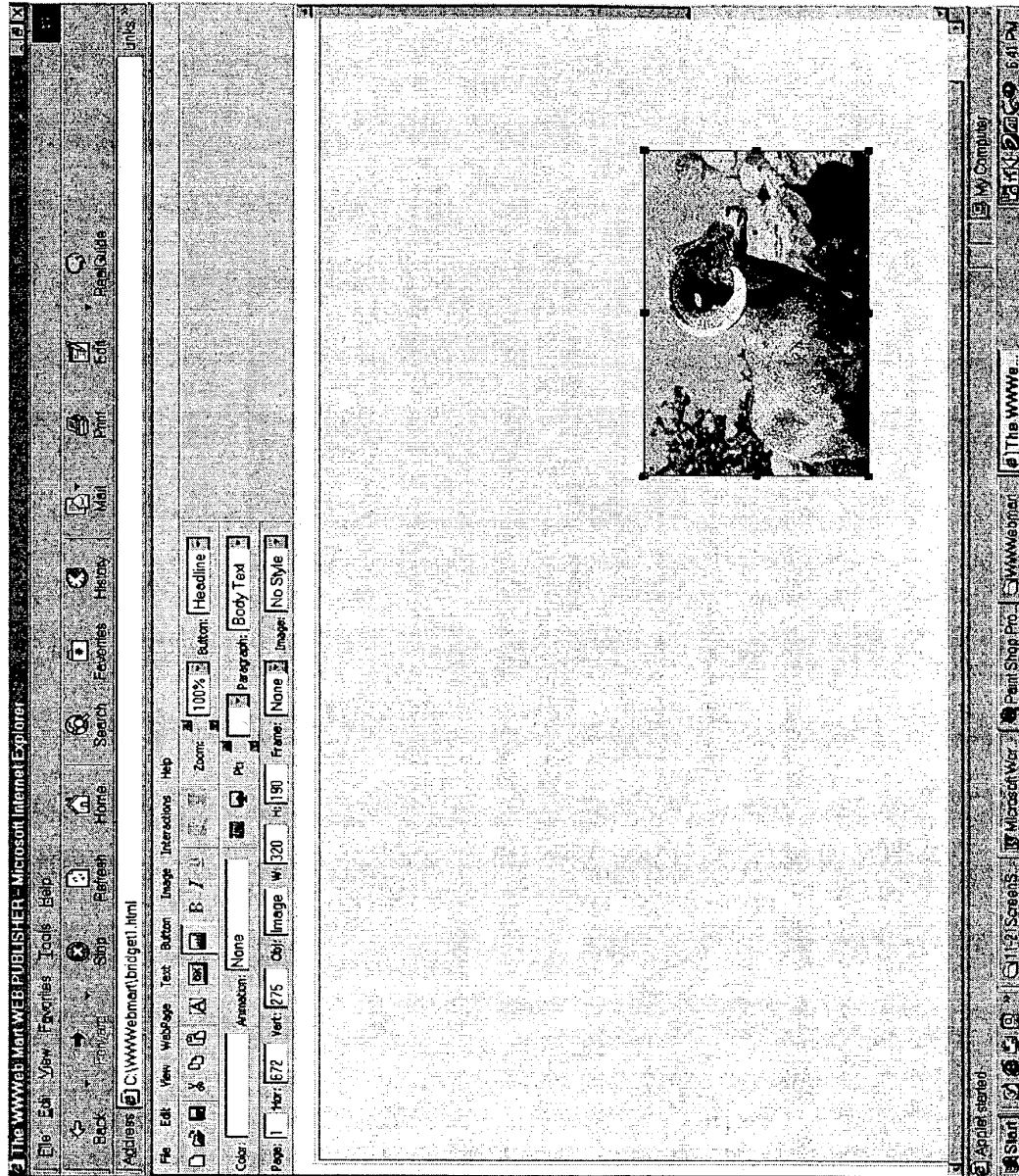


Fig. 53



U.S. Patent

Sep. 22, 2009

Sheet 59 of 68

US 7,594,168 B2

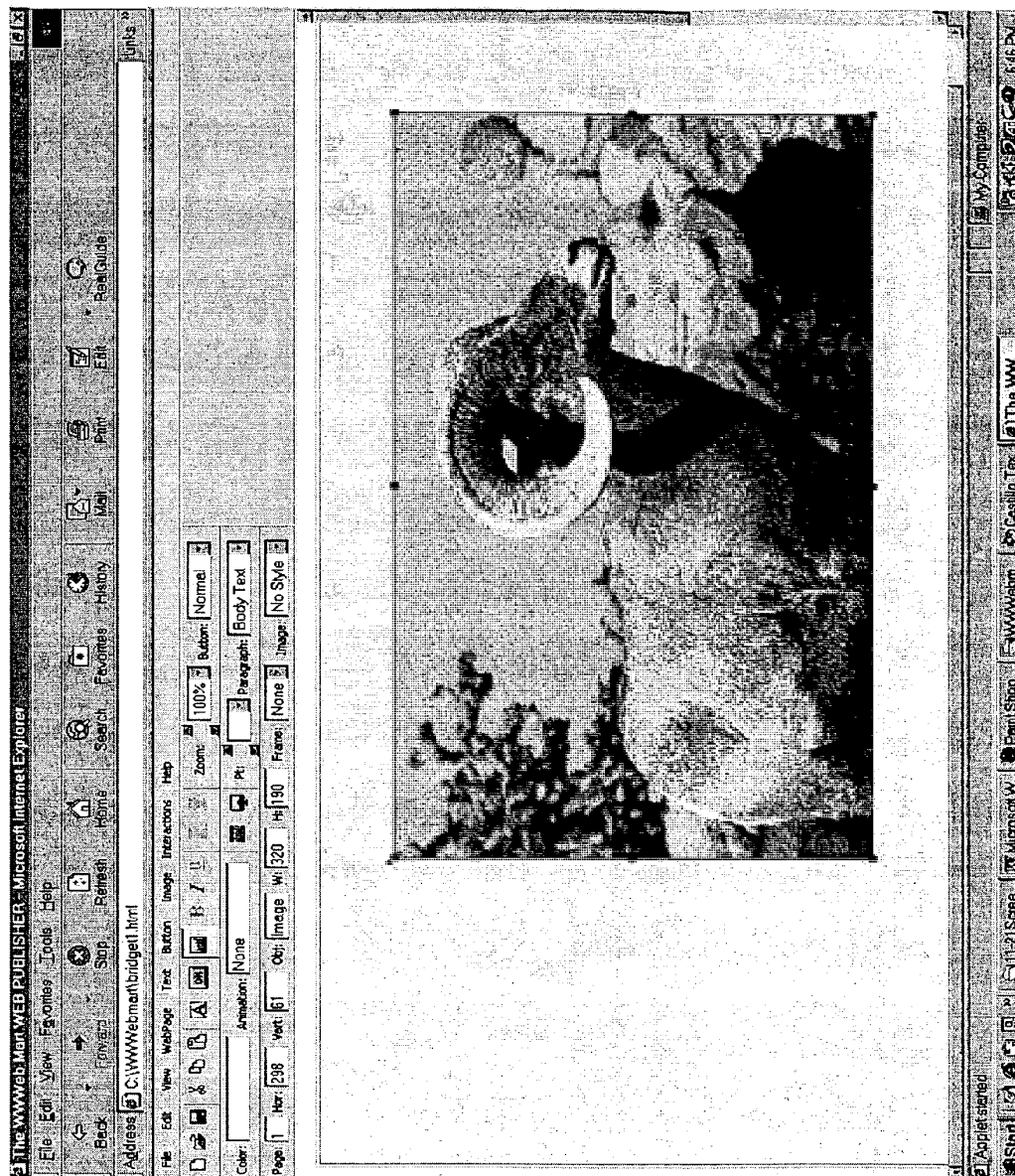


Fig. 54

U.S. Patent

Sep. 22, 2009

Sheet 60 of 68

US 7,594,168 B2

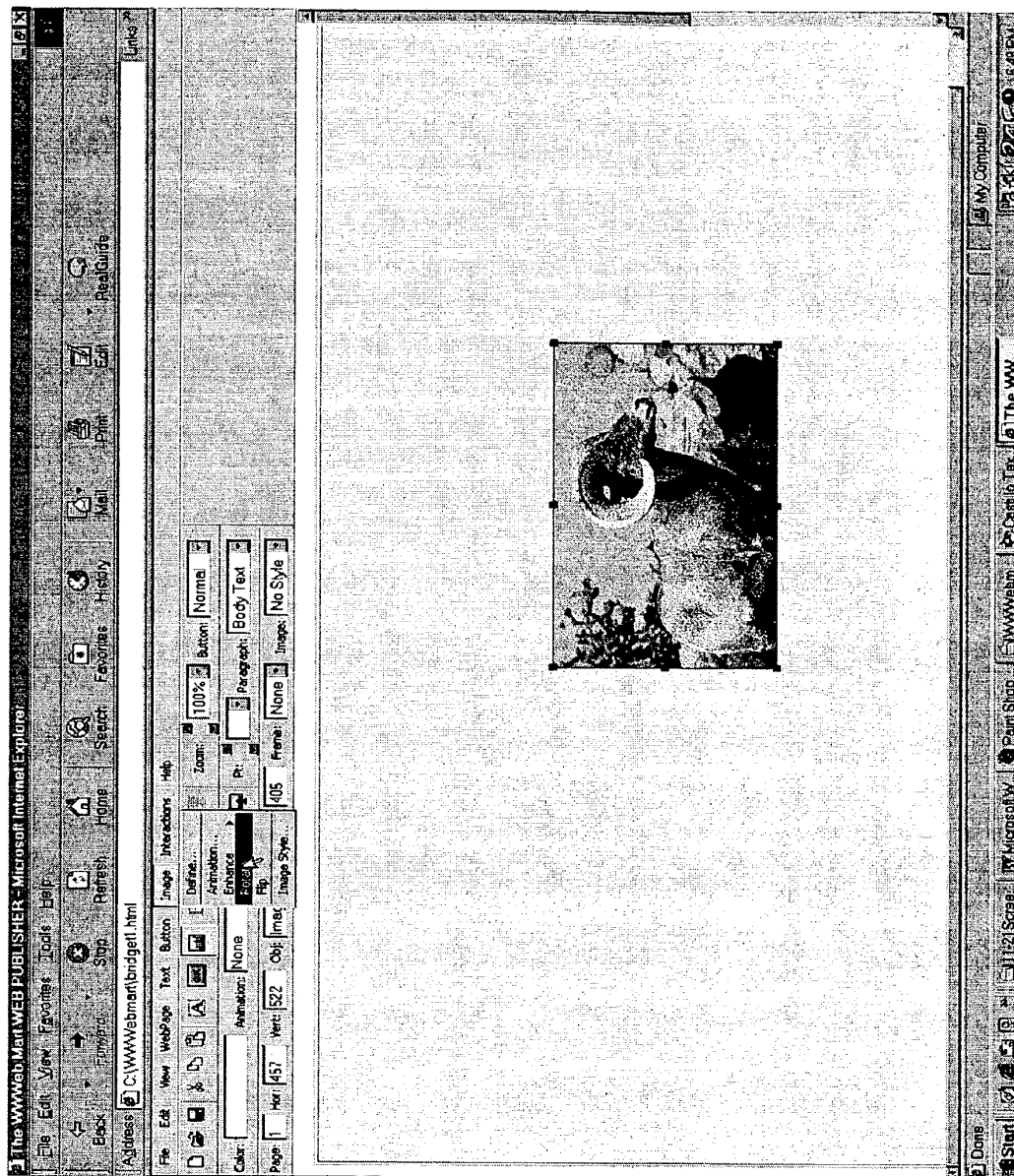


Fig. 55

U.S. Patent

Sep. 22, 2009

Sheet 61 of 68

US 7,594,168 B2

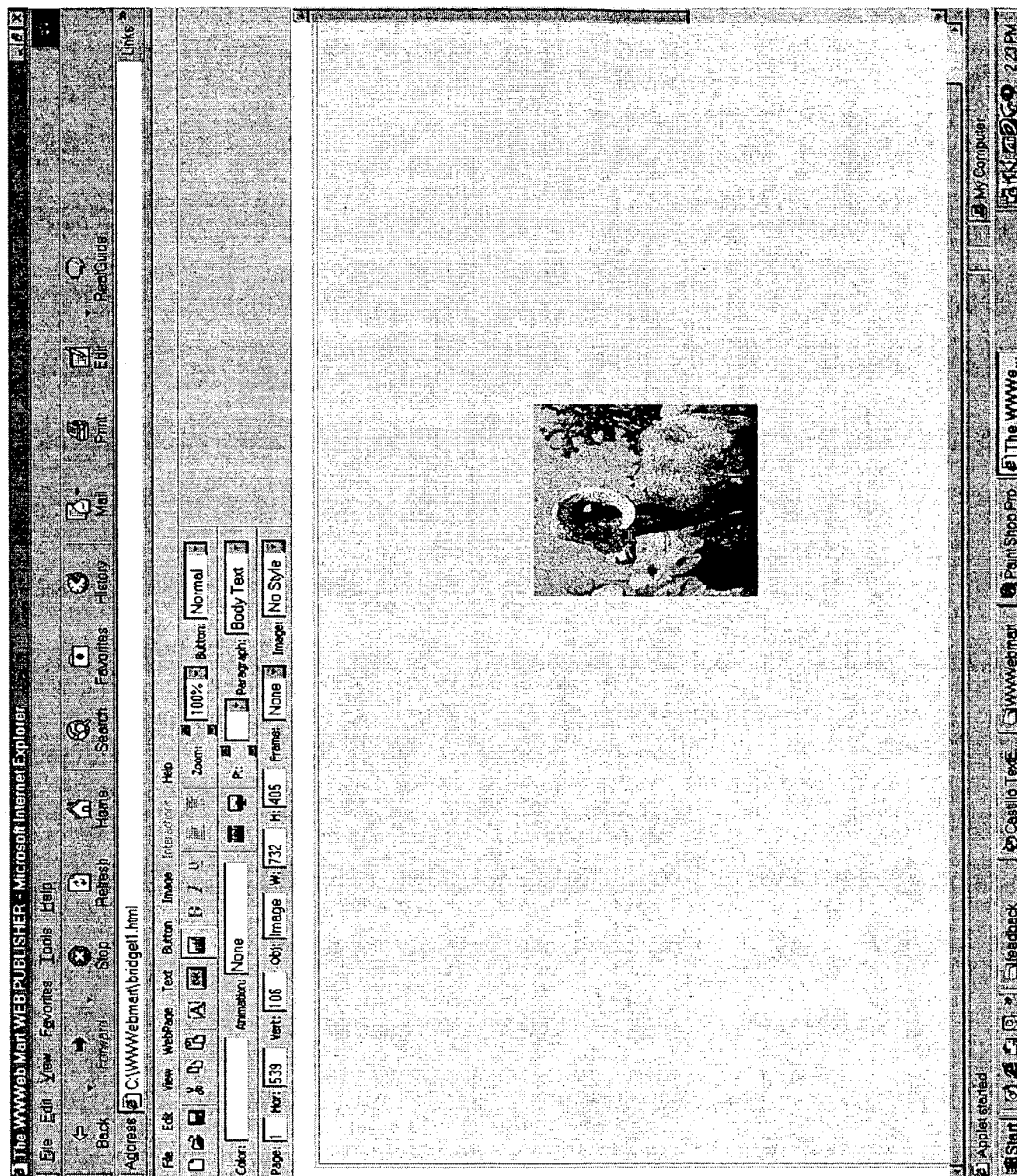


Fig. 56

U.S. Patent

Sep. 22, 2009

Sheet 62 of 68

US 7,594,168 B2

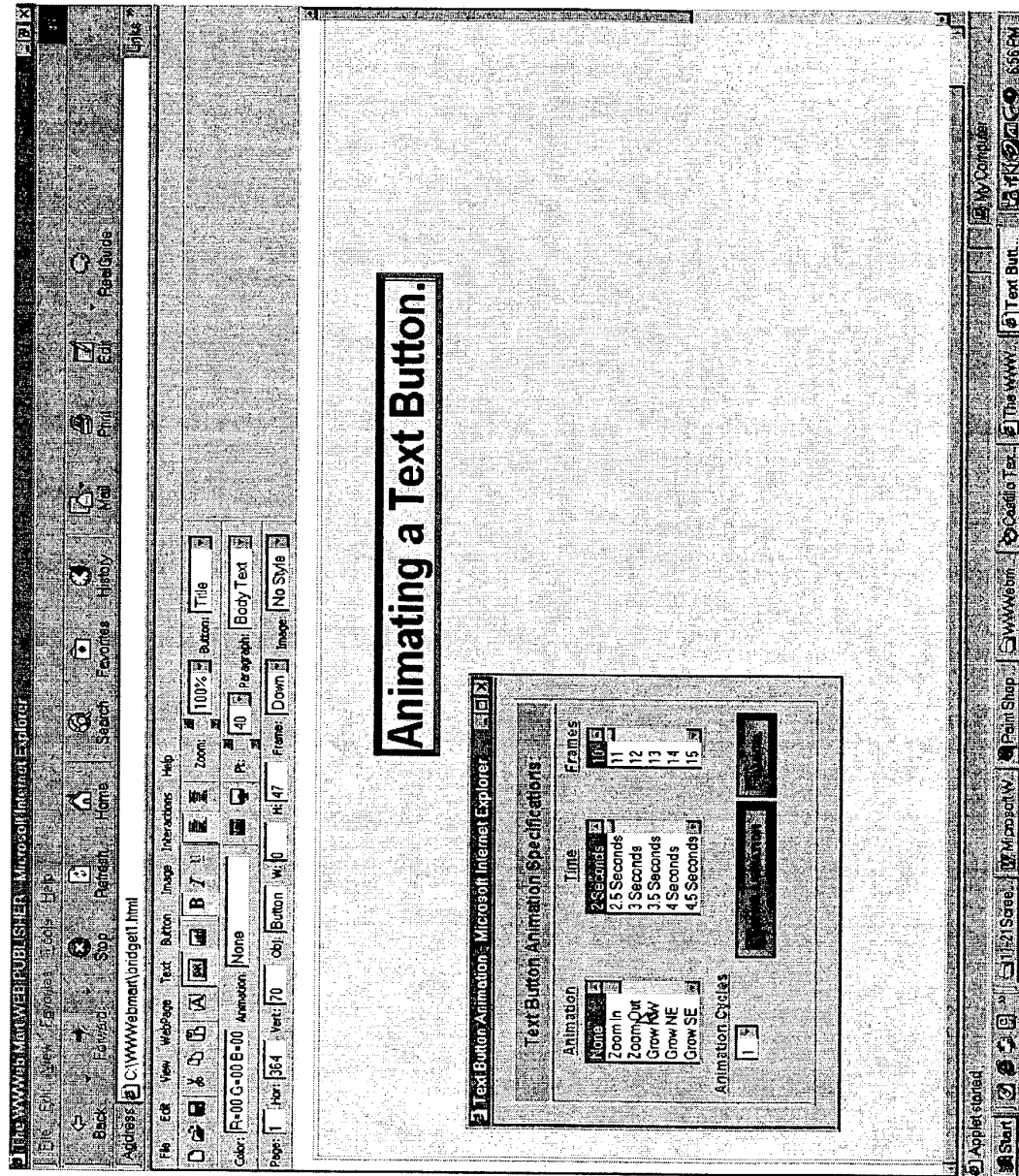


Fig. 57

U.S. Patent

Sep. 22, 2009

Sheet 63 of 68

US 7,594,168 B2

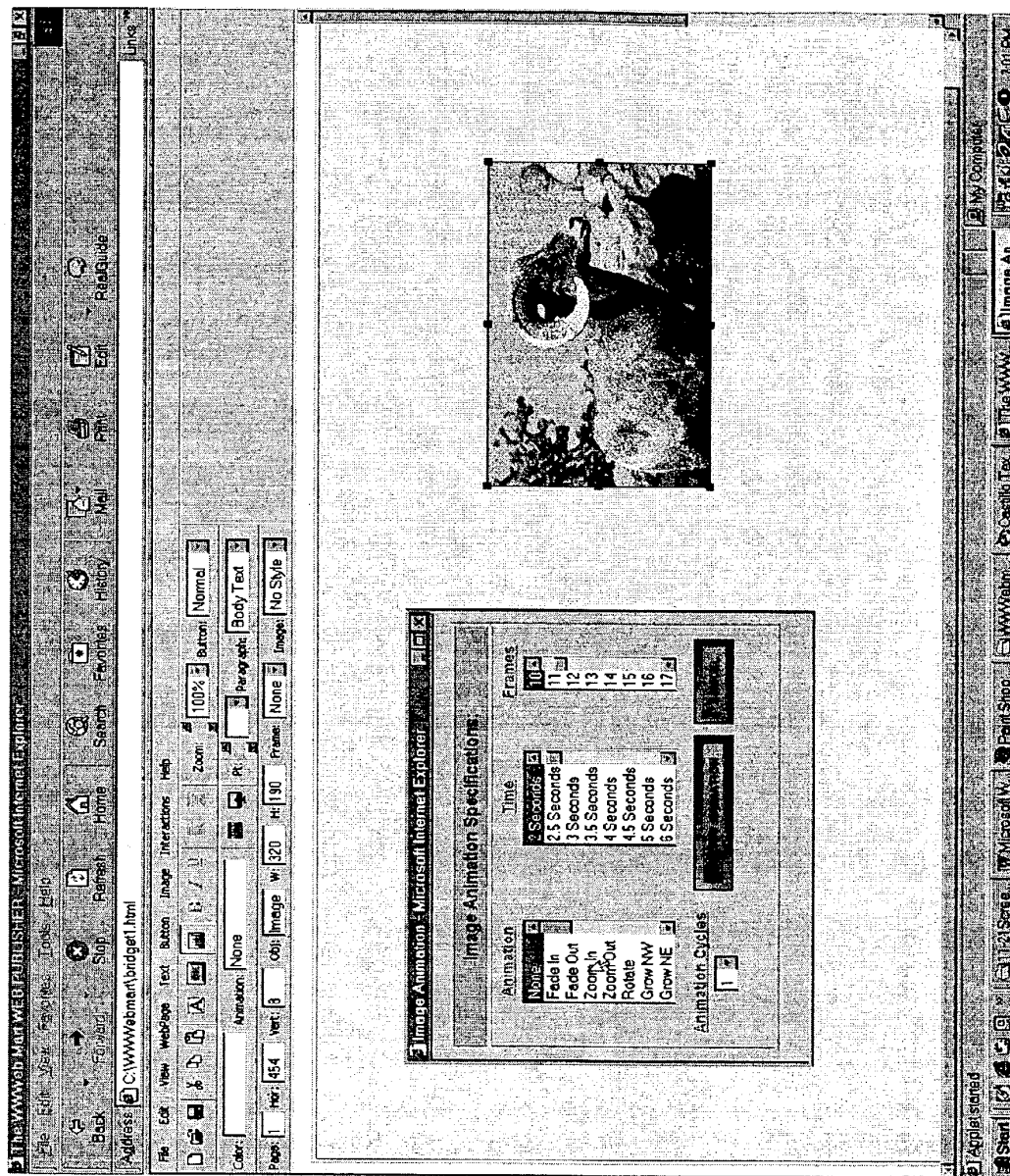


Fig. 58



U.S. Patent

Sep. 22, 2009

Sheet 64 of 68

US 7,594,168 B2

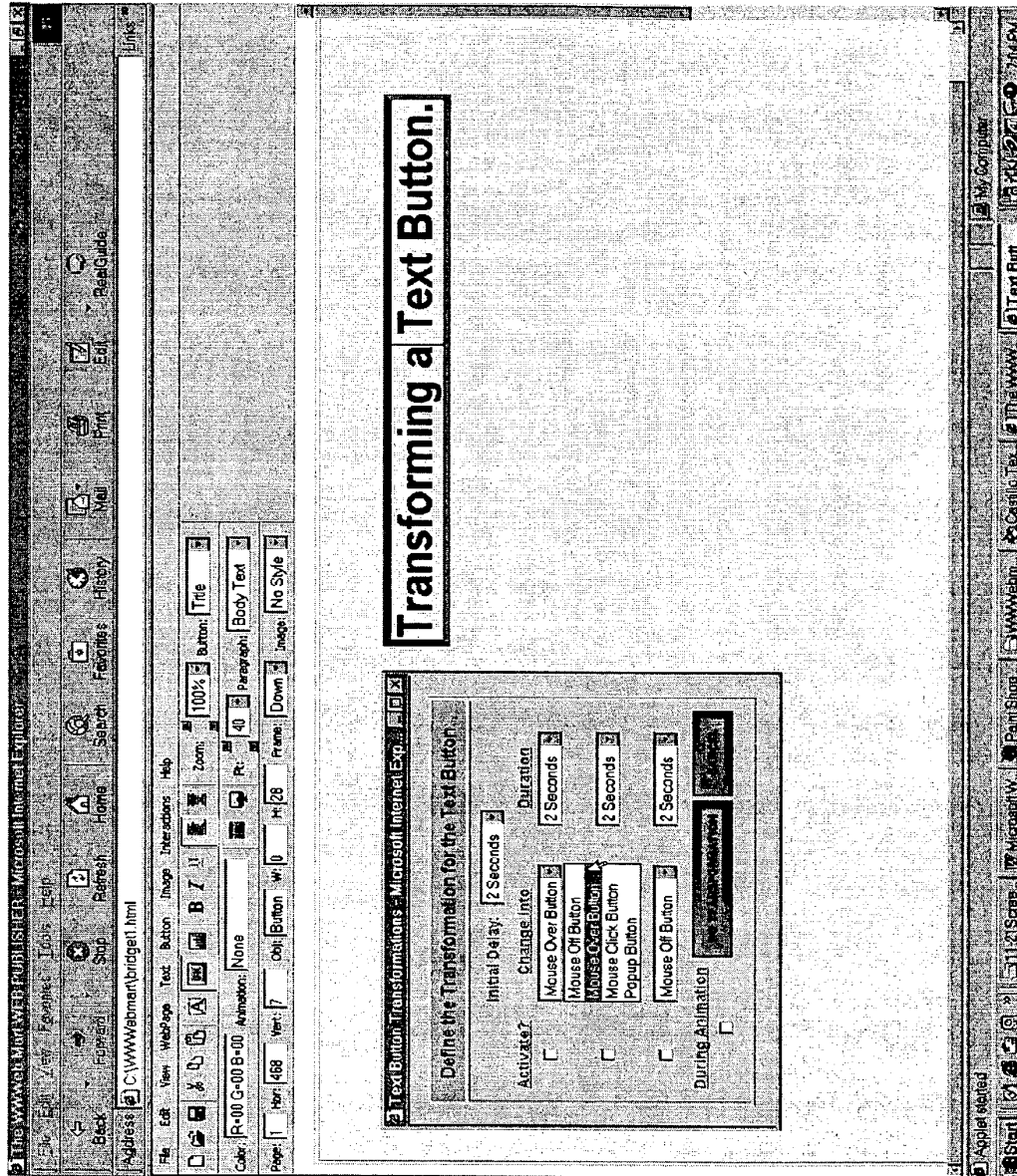


Fig. 59

U.S. Patent

Sep. 22, 2009

Sheet 65 of 68

US 7,594,168 B2

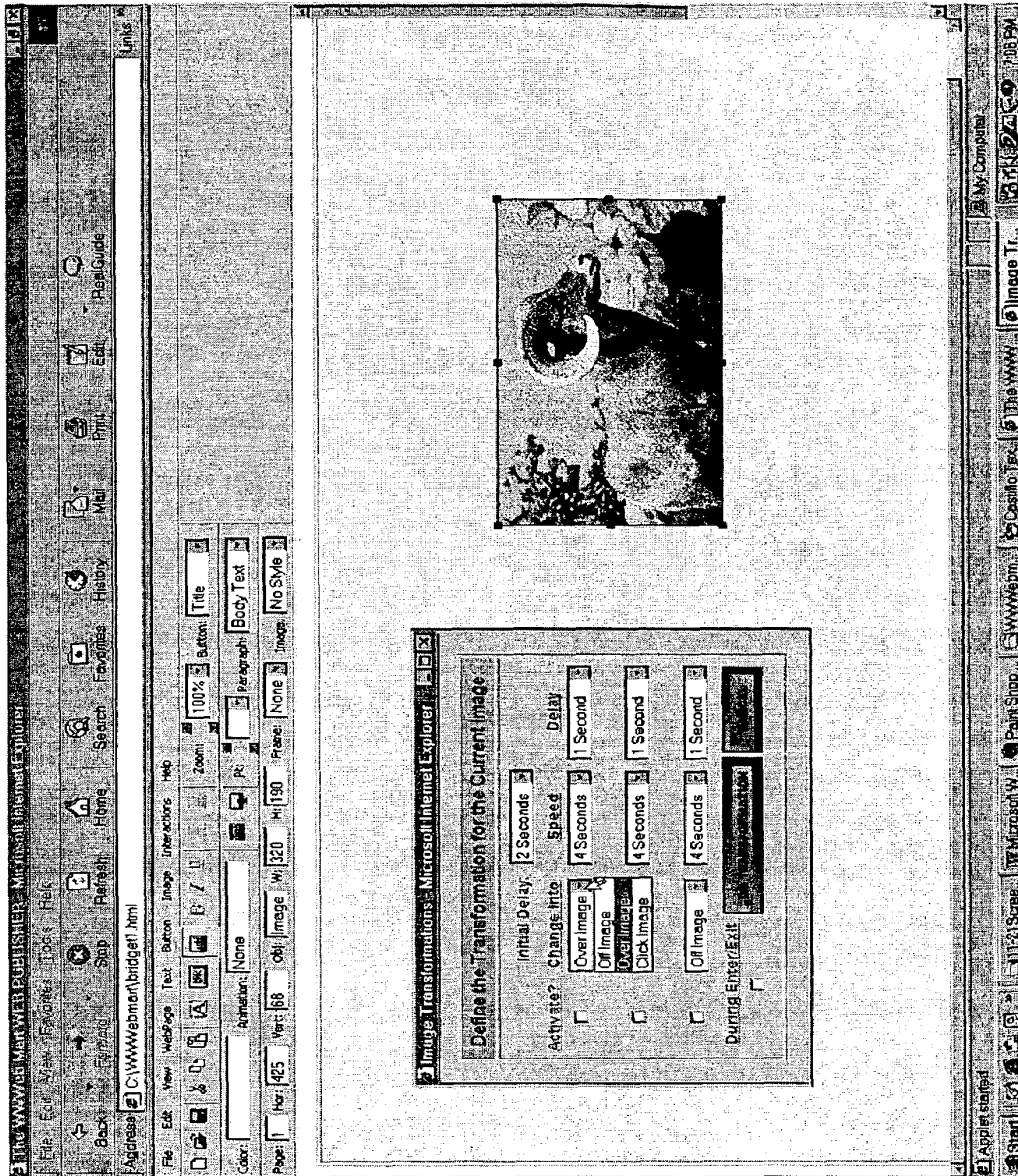


Fig. 60

U.S. Patent

Sep. 22, 2009

Sheet 66 of 68

US 7,594,168 B2

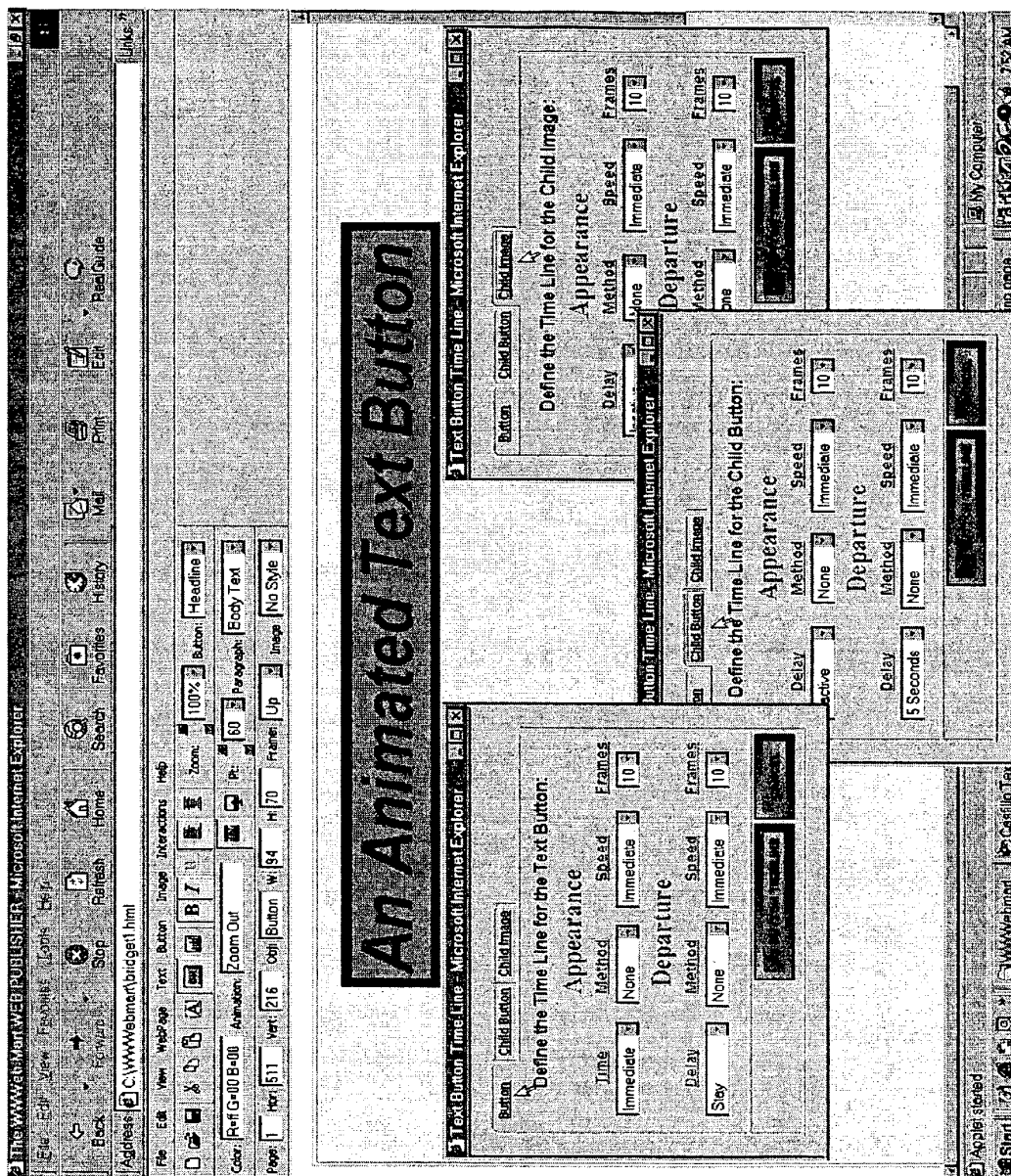


Fig. 61



U.S. Patent

Sep. 22, 2009

Sheet 67 of 68

US 7,594,168 B2

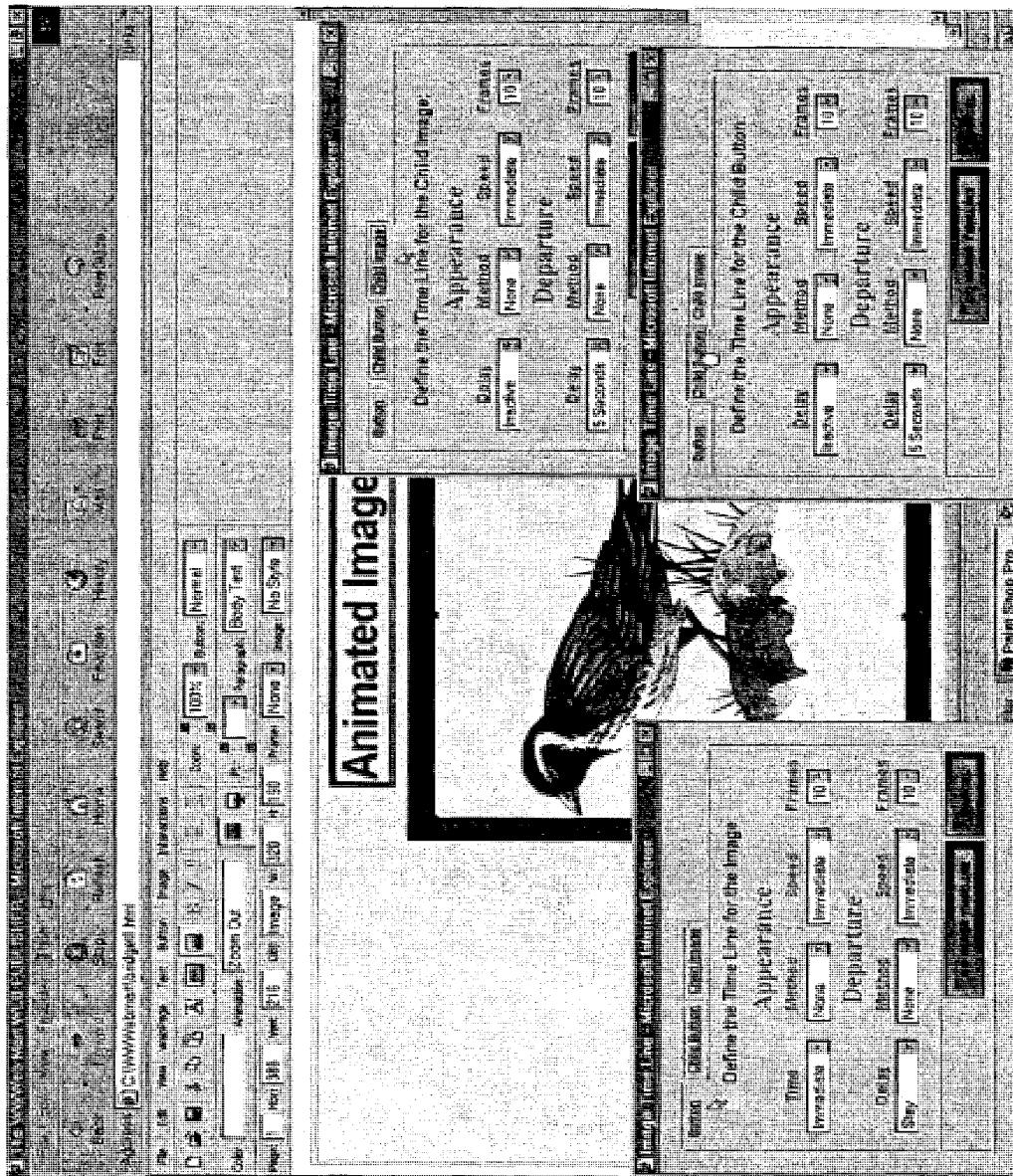


Fig. 62

U.S. Patent

Sep. 22, 2009

Sheet 68 of 68

US 7,594,168 B2

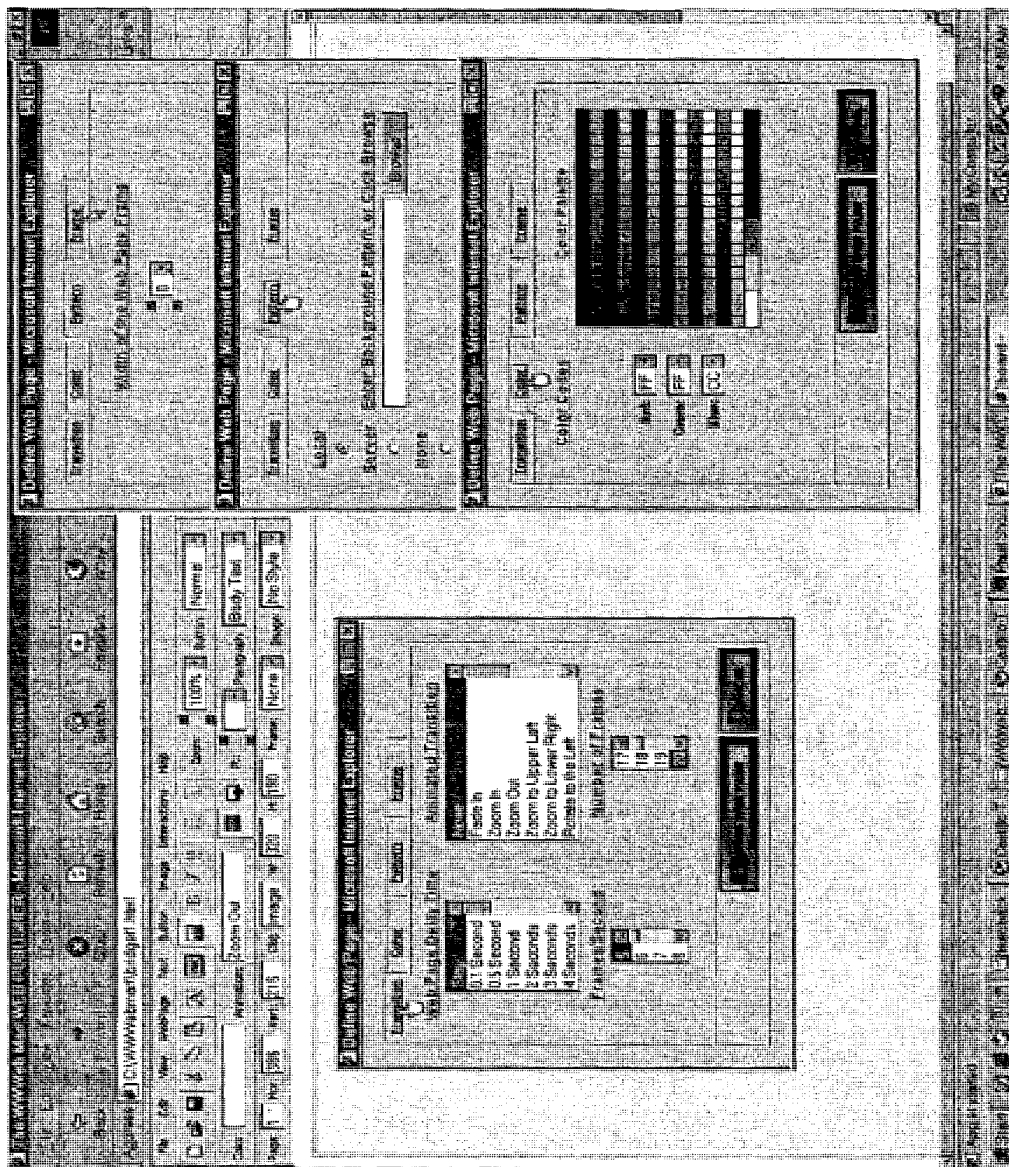


Fig. 63

US 7,594,168 B2

1

**BROWSER BASED WEB SITE GENERATION  
TOOL AND RUN TIME ENGINE****CROSS-REFERENCE TO RELATED  
APPLICATIONS**

This application is a continuation of U.S. patent application Ser. No. 09/454,061, filed Dec. 2, 1999 now U.S. Pat. No. 6,546,397.

**FIELD OF THE INVENTION**

The present application is directed to computing systems, and more particularly to methods and apparatus for building a web site using a browser-based build engine.

**BACKGROUND**

Conventional web site construction tools operate on traditional operating system platforms and generate as output HTML (hyper text mark-up language) and Script Code (e.g., JavaScript). A browser draws a web page associated with the web site by interpreting the HTML and JavaScript Code. However, conventional mark-up and scripting languages include numerous inherent limitations. Conventional mark-up and scripting languages have not been designed for serious multimedia applications. They have almost no file handling ability and very little computational power. In addition, they are remarkably slow and inefficient.

As such it is virtually impossible to write a web publishing application in HTML and JavaScript. All conventional implementations must, and do, utilize a full-featured programming language, such as C++ or Visual Basic. Since the current popular browsers do not support these languages, by necessity, conventional web publishing applications run on platforms other than the World Wide Web (WWW) and its browsers. Therefore, at best, a conventional web publishing application can offer only a crude preview capability of what a real web page will look like.

Although C++ and Visual Basic are very capable languages, the conventional web publishing applications written in these languages are still necessarily limited by the limitations inherent in their form of output, which as described above is typically HTML and scripting code. As such, a conventional web publishing application written in one of these languages suffers from the severe performance problems inherent in these languages.

For example, HTML and JavaScript are incapable of reformatting text and scaling buttons or images dynamically. In addition, most conventional web publishing applications design a web page layout to fit into a 640 pixel wide screen. This means that the ability for higher resolution screens to display more data horizontally is lost. Since capability is wasted on the horizontal plane, unnecessary vertical scrolling may be required. Further, on higher output resolution devices (screens), unsightly extra white space or background may be prevalent.

**SUMMARY**

In one aspect the invention includes a Browser Based build engine that is written entirely in a web based full featured programming language (e.g., JAVA). A Browser Based Interface (the Interface") between the web designer and the build engine is included. The browser-based interface can be written in the World Wide Web's (WWW) Hypertext Markup Language (HTML) and its Extensions (Dynamic HTML,

2

JavaScript and Cascading Style Sheets). The Interface includes a unique set of communication techniques. One technique allows for effective two-way communications between a JAVA engine and JavaScript. Another technique allows for communications between a JAVA applet object inside a JavaScript window, with the JAVA engine, which permits the implementation of advanced intelligent interface objects, such as a "slider" or a "dial".

In one aspect the invention includes a screen resolution sensing mechanism that causes a build engine (i.e. build tools) to adopt its interface to the web designer's screen resolution.

In one aspect, the invention includes a multi-dimensional array structured database, that, in addition to storing the numeric and string data found in conventional databases, also stores multi-dimensional arrays of various multimedia objects. They include colors, fonts, images, audio clips, video clips, text areas, URLs and thread objects. The invention includes a run time generation procedure that creates a compressed web site specific customized run time engine program file, with its associated database and a build engine generated HTML shell file.

The invention can include web page scaling technology, so that when the web site/web page is accessed on the WWW, the web pages and all the objects within them can be scaled to the user's screen resolution and to the then current browser window size.

In one aspect, the invention includes a proprietary multi-level program animation model (threads) that responds to multiple user interactions and time sensitive operations simultaneously.

In one aspect, the invention includes a mechanism for the dynamic resizing of the build engine's web page size during various editing operations.

In one aspect, the invention includes techniques for creating browser based interface objects that visually and behaviorally are identical to those of the MS Window's standard.

Aspects of the invention can include one or more of the following features. A browser based build engine is provided that includes a browser based interface. The entire web site build process is WYSIWYG (what you see is what you get), with the web designer working directly on and with the final web page. The data produced by the build engine is processed and ultimately placed into a multi-dimensional array structured database, and stored in an external file. A run time generation procedure creates a compressed program customized run time engine file, with an associated database and a build engine generated HTML Shell File.

When the web site/web page is accessed on the WWW, web page scaling technology can be accessed to generate web pages that are scaled to the user's screen resolution. A technique is provided so that an applet's size (height and width) can be set in real time under the control of either the interface or the build engine. At the same time a multi-level program animation model (threads) is activated for user interactions and time sensitive operations.

The browser based interface technologies create a set of interface objects with a look and feel that is identical to that of MS Windows, yet includes technologies that equal or occasionally surpass those of high end word processors, desk top publishers, and image processing software programs, particularly in the areas of interaction, animation, and timeline technologies. The run time engine includes multimedia capabilities often rivaling the digital processing capabilities seen on television and in the movies.

Because of the implementation of a variety of performance and file reduction techniques, the entire run time environment

US 7,594,168 B2

3

can range from as low as 12K, and no larger than 50K. This depends upon the features selected by the web designer. Although the compressed image, audio, and/or video files must also be downloaded, with a reasonable web site design, web pages should load quickly. The initial run time environment is no larger than 25K, thus the initial web page should generally load in less than 2 seconds, and subsequent web pages in less than 1 second with a 56K modem, even with numerous image files.

The present invention provides a real time, dynamic linkage between JAVA and HTML including two-way communications, in real time, between JAVA and JavaScript.

#### BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing aspects and many of the attendant advantages of the invention will become more readily appreciated through the following drawings and their associated screen shots, referred to throughout the detailed description, wherein:

FIG. 1 is a flow chart depicting a prior art conceptual overview of how a user and a web browser interface.

FIG. 2 is flow chart depicting a conceptual overview of how a user interfaces with a web browser when implementing the present invention to construct a web site.

FIG. 3a is a schematic diagram showing the main components of a build tool in accordance with one implementation of the present invention.

FIG. 3b is a process flow diagram showing a build process in accordance with one implementation of the present invention.

FIG. 4a is schematic diagram showing the main components of a run generation tool in accordance with one implementation of the present invention.

FIG. 4b is process flow diagram showing a run time process in accordance with one implementation of the present invention.

FIG. 5 is a flow chart, with its attendant screen shot shown in FIG. 37, that depicts a detailed view of a build time initialization procedure in accordance with one implementation of the present invention.

FIG. 6 is a flow chart, with its attendant screenshots shown in FIGS. 38-48, that depicts a detailed view of the build time supported user input techniques and techniques for communication of data and status between the build engine and the interface in accordance with one implementation of the present invention.

FIG. 7a is a flow chart that shows an overview of the build time techniques for implementation of pop-up windows (usually called "dialog boxes" in MS Windows), the panel interface, and interface for color selection.

FIG. 7b is a flow chart, with its attendant screenshots shown in FIGS. 37-38, that shows a detailed view of the build time techniques for implementation of panel interface objects, including the menu bar, menus and sub-menus, the tool bars, status fields, interactive fields, and interactive pull down lists, in accordance with one implementation of the present invention.

FIG. 7c is a flow chart, with its attendant screenshots shown in FIG. 37 and FIG. 63, that shows a detailed view of the build time techniques for implementation of tabbed pop-up windows (also called "dialog boxes" in MS Windows).

FIG. 8 is a flow chart that shows a detailed view of the build time techniques for updating the internal databases and the setting of feature flags for run time optimization purposes.

4

FIG. 9 is a flow chart, with its attendant screenshot shown in FIG. 37, that shows a detailed view of the build time polling methods used to facilitate communication from the JAVA build engine to the interface.

FIG. 10 is a flow chart that shows a detailed view of the build time techniques for analyzing user input for error checking and data integrity.

FIG. 11 is a flow chart, with its attendant screenshot shown in FIGS. 38-41, that shows a detailed view of the build time methods for direct text entry at an arbitrary cursor position and text editor implementation methods.

FIG. 12 is a flow chart, with its attendant screenshot shown in FIGS. 49-56, that shows a detailed view of the build time techniques for reading external image files, creating them on a web page, and then manipulating them through either direct mouse interaction or through the interface's panel/windows.

FIG. 13 is a flow chart that shows a detailed view of the build time implementation of text, button and image styles in accordance with one implementation of the present invention.

FIG. 14 is a flow chart that shows a detailed view of the video and audio processing in accordance with one implementation of the present invention.

FIG. 15 is a flow chart that shows a detailed view of the frame, table, forms, and draw objects processing and technology in accordance with one implementation of the present invention.

FIG. 16 is a flow chart that shows a detailed view of the build time methods for supporting various user interactions at run time.

FIG. 17 is a flow chart, with its attendant screenshots shown in FIGS. 57-58, that shows a detailed view of the build time methods for text button and image object animation.

FIG. 18 is a flow chart, with its attendant screenshots shown in FIGS. 59-60, that shows a detailed view of the build time methods for text button and image transformations.

FIG. 19 is a flow chart, with its attendant screenshots shown in FIGS. 61-62, that shows a detailed view of the build time methods for text button and image time lines.

FIG. 20 is a flow chart with its attendant screenshot shown in FIG. 63, that shows a detailed view of the build time web page transition animations and time lines.

FIG. 21a is a flow chart that shows a detailed view of file operations performed in accordance with one implementation of the present invention.

FIG. 21b is a flow chart that shows a detailed view of the view operations performed in accordance with one implementation of the present invention.

FIG. 22 is a flow chart that shows a detailed view of a dynamic web resizing process that is activated by the "Open" and "Web Site" commands under the "File" menu and the "Zoom" command under the "View" menu.

FIG. 23 is a screen shot showing a file selection window operation in accordance with one implementation of the present invention.

FIG. 24 is a flow chart showing a detailed view of an external database in accordance with one implementation of the present invention and also shows the security and optimization techniques that can be employed.

FIG. 25 is a flow chart showing a detailed view of a method for creating a customized and optimized run time engine in accordance with one implementation of the present invention.

FIG. 26 is a flow chart showing a detailed view of the methods for creating an HTML shell file in accordance with one implementation of the present invention.

FIG. 27 is a flow chart showing a detailed view of the methods for creating compressed CAB and JAR files in accordance with one implementation of the present invention.

US 7,594,168 B2

5

FIG. 28 is a flow chart showing a detailed view of the technology for dynamic web page size creation in accordance with one implementation of the present invention.

FIG. 29 is a flow chart showing a detailed view of the methods for reading the multimedia database and generating the necessary objects in accordance with one implementation of the present invention.

FIG. 30 is a flow chart showing a detailed view of the methods for dynamically scaling the web page object(s) to different screen resolutions and window sizes in accordance with one implementation of the present invention.

FIG. 31 is a flow chart showing a detailed view of the methods for executing a multi-level web page and object thread architecture in accordance with one implementation of the present invention.

FIG. 32 is a schematic diagram that shows a detailed view of the web page transition animation architecture in accordance with one implementation of the present invention.

FIG. 33 is a schematic diagram that shows a detailed view of the parent object time line architecture in accordance with one implementation of the present invention.

FIG. 34 is a schematic diagram that shows a detailed view of the child object time line architecture in accordance with one implementation of the present invention.

FIG. 35 completes the flow chart begun at FIG. 31.

FIG. 36 is a flow chart showing a detailed view of the methods for responding to user interactions in accordance with one implementation of the present invention.

FIGS. 37-63 are screen shots of the user interface presented by the build process in accordance with one implementation of the present invention.

#### DETAILED DESCRIPTION

Referring to FIG. 1, in a prior art process for creating and displaying a web site, the user either directly writes HTML and Script Code providing user input at 1 or operates a related prior art product at 2, which generates the HTML and Script Code at 3. A separate file, with its attendant HTML and Script Code is uploaded for each separate web page in the web site at 4, which is then interpreted by a browser when accessed at 5.

FIG. 2 shows a process for creating and displaying a web site in accordance with one aspect of the invention in which, a user operates a build tool at 6, working directly with one or more of the final web pages in a full WYSIWYG mode. The build tool accepts the user input and creates a multi-dimensional embedded multimedia object database at 7. A run time generation process is then invoked to create the necessary run time files at 8 (including HTML shell, CAB/JAR files and a customized runtime engine) which are then loaded to a user's web site at 9. The web page(s), when viewed by a web surfer, are activated by the browser calling the customized run time engine at 10. The run time engine then begins to read the database and down load image, audio and video files, while simultaneously drawing the first web page for viewing or user interaction at 11.

#### Build Tool and Process

FIG. 3a shows a build tool 350 at the detailed component level. The build tool includes a build engine 352, interface 354, screen sensing mechanism 356, multi-dimensional array structured database 358, interface's database 360, web page scaling engine 364, time line engine 366 and installation Program 368. The operation and use of each of these components is described in greater detail below.

6

FIG. 3b is a flow of the build process executed by the build tool to create a web page/web site. Referring to FIGS. 3a and 3b, the process begins with an initialization (12) and continues through to a point where a web site has been defined and stored in the build engine's internal database (29).

The build tool 350 includes plural individual tools that are created and initialized at (12). The processes for creating and initializing build tools are described in greater detail below in association with FIG. 5. After the build tools are created and initialized at 12, the build tool 350 interacts with the user, receiving user commands (actions), for example, to build a web site. The build tool 350 processes user responses and communicates the same and status information to both the build engine 352 and interface 354 at 13. The processes for interacting with the user are described in greater detail below in association with FIG. 6.

In one implementation, the interface includes a panel (and its objects, including a menu bar, menus and sub-menus, tool bars, status fields, interactive fields and interactive pull down lists), pop-up windows (called "dialog boxes" in MS Windows), color and alert message interface technologies, built with HTML, Dynamic HTML (DHTML), JavaScript, and Cascading Style Sheets (CSS). Interface 354 responds to the user input and may display a pop-up window, update the interface objects, or display alert messages, as shown at 15. The operation of the interface 354 is described in greater detail below in association with FIG. 7a, FIG. 7b and FIG. 7c.

As the build engine 352 receives data and status information, it updates an internal database (part of multi-dimensional array structured database 358) and sets feature flags at 14. The processes for updating the internal database and setting flags are described in greater detail below in association with FIG. 8. To enable effective two-way communication between the interface and the build engine, polling technology is included as shown at 16. The details of the polling process are described in greater detail below in association with FIG. 9.

Whenever user input is received, the build tool 350 analyzes the input including error checking at 17. In one implementation, the input is analyzed and then processed by object type (class). The process for analyzing input to determine type is described in greater detail below in association with FIG. 10. In one implementation, the number of different object processing technology classes are four, and include direct text entry (18), image processing (19), video or audio files and links (21) and frames, tables, forms and draw objects (22). The build tool 350 processes the user input based on class. The processes invoked for direct text entry are described in greater detail below in association with FIG. 11. The processes invoked for image processing is described in greater detail below in association with FIG. 12. The processes invoked by the text button, paragraph, and image style technologies are described in greater detail below in association with FIG. 13. The processes invoked for processing audio and video files and channels are described in greater detail below in association with FIG. 14. The processes invoked for processing frames, tables, forms and draw objects are described in greater detail below in association with FIG. 15. When an image, text button or paragraph object is to be inserted in the web page, the current style that is selected in the panel defines the initial settings used when creating the object in the web page. As such, button, image and paragraph style setting and technology will be invoked at 20 depending on the user input. The processes invoked by the paragraph style setting and technology is described in greater detail below in association with FIG. 13.

US 7,594,168 B2

7

After the input is processed as described above, a check is made to determine if one or more animation or transformation (interaction) techniques are to be invoked at 23. The run time engine provided in accordance with the teachings of the present invention support various user interactions, including support for numerous animation and transformation techniques, and both web page and object time lines. Depending on the user selections, one or more technologies may be invoked. In the implementation shown, the build tool 350 is configured to check to determine if the input data is related to plural technologies including: user interaction technology (24), animation technology (25), transformation technology (26), object time line technology (27) and web page transition animation technology (28). The processes invoked for user interaction technology are described in greater detail below in association with FIG. 16. The processes invoked for animation technologies are described in greater detail below in association with FIG. 17. The processes invoked for transformation technologies are described in greater detail below in association with FIG. 18. The processes invoked for object timeline technologies are described in greater detail below in association with FIG. 19. The processes invoked for web page transition animation technologies are described in greater detail below in association with FIG. 20.

After the build tool 350 has processed the user input, one or more file operations can be invoked at 29a. In one implementation, the file operations are "save", "save as", "new", "close", "open", "apply" and "web site". If "open" or "web site" are selected, the build tool 350 initiates the dynamic web page resizing process at 29c (See FIG. 22). If "save" or "save as" are selected, the build tool 350 initiates a run generation process (See FIG. 4 and FIG. 24). File operations "close", "open", and "new" can also initiate the run generation process, based on the state of the build process and user action.

At any time during the processing of user input, one or more view operations can be invoked at 29b. In one implementation, the view operations supported are "normal", "preview", "play", and "zoom" (at various zoom percentages). If any of the "zoom" levels are selected, the build tool initiates the dynamic web page resizing process at 29c (See FIG. 22). If the "preview" or "play" view operations are selected they will initiate the run time process (See FIGS. 28 through 36).

FIG. 4a shows a run generation and runtime tool 370 at the detailed component level. The run generation and runtime tool 370 includes a run generation procedure 371, web scaling engine 372, a database 374 and a (web) page size generation engine 376 and run time engine 377 including a runtime user interaction engine 378, a runtime timeline engine 380 and a runtime drawing, animation, audio, and video engine 382. In one implementation, run time engine 377 includes plural engines, each of which may in themselves include plural engines.

FIG. 4b shows the run processes including methods for creating the run time files, including the external database, the web site specific customized run time engine, the HTML shell file, and the compressed CAB/JAR file. The run processes also include methods for scaling each web page to the web surfer's then current screen resolution and web browser window size. After a web page has been scaled, a run time engine executes a multi-level thread technology, which presents to the viewer web pages that can operate under time lines that may include animated transitions. Associated with the web page time lines can be object time lines that may define entrance, main and exit animations, transformations, and synchronized time lines for child objects. Each object can have

8

multiple object states, responsive to various user interactions, which can result in numerous types of visual and audio responses and actions.

Referring now to FIGS. 4a and 4b, a run generation process 360 begins by invoking the run generation procedure 377. The run generation procedure 371 begins by creating the external database (part of database 374) at 30. The external database may include references to image, video and audio files, and video and audio channels. The process for creating the external database is described in greater detail below in association with FIG. 24. A customized and optimized run time engine (run time engine 377) is created at 31. The customized and optimized run time engine (run time engine 377) generates the web pages for the web site and is activated from the user's server. The process for creating the run time engine 377 is described in greater detail below in association with FIG. 25. The HTML shell file is created at 32, and then the CAB and JAR files are created at 33a. The HTML shell file includes JavaScript Code to activate and interrogate the page size generation engine 376, and to activate the entire runtime engine. The CAB and JAR files both include the runtime engine and database in compressed executable form. The CAB file(s) will be activated by the HTML shell file if it senses the browser as being Microsoft Explorer, otherwise it will activate the JAR file(s). The processes for creating the HTML shell file and the CAB and JAR files are described in greater detail below in association with FIG. 26 and FIG. 27, respectively. The run generation process portion of the run processes is completed as the HTML shell file and the CAB and JAR files are uploaded to the user's web site at 33b.

After the upload, the run time process 365 portion begins with the run time engine 377 invoking a web page size generation technology (engine) 376 at 34. The web page size generation technology can be used to determine the screen resolution and the current browser window size. The process for invoking and initializing the web page size generation technology is described in greater detail below in association with FIG. 28. The external database is read and the necessary objects generated at 35 from their stored external references. These objects include image, audio, and video objects. The processes for generating the necessary objects are described in greater detail below in association with FIG. 29. A web page generation and scaling technology (web page scaling engine 372) is then invoked at 36. The web page scaling engine 372 can be used to reformat and scale objects that had been placed in a web page during the build process. The processes employed by the web page generation and scaling technology are described in greater detail below in association with FIG. 30. The run time engine then, as necessary, executes a multilevel web page and object thread technology at 37 while the runtime user interaction portion 378 of run time engine 371 responds to user interactions at 38. The processes invoked by the multilevel web page and object thread technology are described in greater detail below in association with FIGS. 31-35. The processes invoked by the run time engine to respond to user interactions are described in greater detail below in association with FIG. 36.

#### Detailed build processes

Referring now to FIGS. 3a and 5 through FIG. 22 the build tool 350 and its associated build process are described. Referring first to FIGS. 3a and 5, initialization methods are shown. At 39 the build tools are created as part of the execution of the installation program 368. They can include:

- 1: Initial build tool HTML/JavaScript file (IBTF)
- 2: An initialization engine (IE).
- 3: A build engine.



US 7,594,168 B2

9

- 4: The build engine parent HTML frame file. (PFF).
- 5: A "Control Panel and Status Line" HTML/JavaScript File ("panel") for;
  - Controlling the JavaScript database.
  - Calling and initializing all pop-up windows.
  - Reading all pop-up window values, and updating a JavaScript database
  - Calling the build engine and passing all necessary data and status information.
  - Polling the build engine for two-way JAVA/JavaScript communication.
  - Displaying and updating the status of its interface objects.
  - Issuing alert messages.
  - Processing direct user interactions with the panel's interface objects.
- 6: Numerous HTML/JavaScript files, one for each pop-up window.
- 7: JAVA applets, embedded in HTML/JavaScript pop-up window files.
- 8: A build engine HTML definition file that is created and modified dynamically.

d The initialization and build engines can be placed in a JAVA wrapper so that JavaScript code may receive and process return values from JAVA methods. The initialization and build engines are also created in a "Signed" CAB file, and assigned the necessary security rights, so that the engines can assert the necessary permissions, if permitted by a given browser's security manager, when read or write operations are required. In one implementation, an installation program is run prior to the first use of the build tools. After installing all of the files, the installation program can install the necessary class libraries required by the run generation process in which the customized and optimized run time engine is created (See FIG. 25). The installation program can also set the necessary environmental variables and installation options.

At 40 the web surfer points a browser at (i.e. calls) an initial build tool HTML/JavaScript file (IBTF). At 41 the IBTF identifies the current browser type and version number. Presently, each browser has different security manager implementations. In one implementation, the invention supports the following three categories:

- 1: With appropriate signing and time stamping, and with appropriate assertions of permissions, the browser will permit local read/write operations.
- 2: With appropriate signing and time stamping, and with appropriate assertions of permissions, the browser will permit local read operations, but write is only legal if sent to a server.
- 3: Local read/write operations are illegal, but are permitted on the server.

The IBTF can include a flag that can be set to indicate which security implementation is supported, so that all subsequent read/write operations will comply with the current browser's security manager.

At 42, the IBTF causes the browser to execute the IE so as to sense the screen resolution and for adapting the interface to the user's screen resolution. In one implementation, after entering a delay loop and waiting for the IE to report it is fully loaded and initialized, the IBTF calls two IE methods, which return the width and height of the current screen and browser window. The IBTF then checks for the presence and value of a "mode cookie", to determine whether this is an initialization process, a web site open command process, or a dynamic web page resizing process. If the mode cookie is set to initialize, or it doesn't exist, the IBTF calls the IE to generate the build

10

engine's HTML definition file. At 43 the IE then asserts the required security permission and at 44 creates a build engine HTML definition file and writes this file to the local disk (as appropriate). At 45 the IBTF then turns control over to the PFF for activating the "panel" and build engine and displaying the build engine user interface screen.

The build engine user interface screen includes a "panel" portion and a build engine portion, each of which are loaded into their respective frames, after which the web site page(s) build process can begin. Screen shot FIG. 37 shows a representation of the user interface presented by the build tool. The user interface includes a panel 400 and build frame 500. Panel 400 includes a menu bar 410, menus 420 and sub-menus 430, tool bars 440, status fields 450, interactive fields 460, interactive pull down lists 470 and operational pop-up windows 480. The menu bar 410 can be used for selecting a menu command that will cause a menu to be drawn. The menu (one or menus 420) can be used to select a feature command that could cause an operational pop-up window to be drawn, a direct user input technique or object manipulation technique to be activated, or a sub-menu 430 to be drawn. A sub-menu (one of sub-menu 430) can cause the same type of events as that of a menu. The tool bars 440 include various icons that are shortcuts to feature commands that are also available through the menu bar and its menus. In addition, the tool bar 440 can be used to show the current state of a feature. Status fields 450 show the current value of a certain setting. Interactive fields 460 also show the current value of a setting, but can also be directly changed by the user by typing into the field, with the result immediately processed by the build engine 352 and displayed in the build frame 500. Interactive pull-down lists 470 also show the current value of a setting, but, if selected with a mouse click, will drop down a selection list, which may have an elevator attached. The user can click on an item in the selection list, which will become the current setting with the result immediately processed by the build engine 352 and displayed in the build frame. Operational pop-up windows 480 can have tabs assigned if the number of choices within the pop-up window is large. One or more settings can be changed through a pop-up window, with the results immediately processed by the build engine 352 and displayed in the build frame 500. These interface techniques are described in greater detail below in the build process.

The build frame 500 is used to present the actual web page as constructed by a user. The user can directly enter text, import images, video and audio for display/playback and create animations and transformations that can be viewed in the build frame. FIG. 6, with its attendant screen shots FIGS. 38 through 48, shows the user input techniques supported in one implementation of the invention. In one implementation, the user inputs supported include: selection from a JAVA window object (48); selection from a JavaScript window (49) including selection with dual spin control (50a) or selection from a JavaScript child window object (50b); direct text entry (51); page resizing (52); direct object manipulation (53); and, selection from a JavaScript panel (54).

In the implementation shown, of the six user input techniques sensed at 13, the code for supporting selections from a JavaScript pop-up window at 49 and selections from the "panel" at 54 were implemented entirely in HTML/JavaScript Code, while support for direct text entry at 51 and direct web page object manipulation at 53 were implemented entirely in JAVA (or any other browser-based full featured programming language). In one implementation, code for supporting selections from a JAVA Window object at 48 and dynamic web page resizing at 52 are implemented using both HTML/JavaScript and JAVA. Those of ordinary skill will

US 7,594,168 B2

11

recognize that, JAVA could have been used more extensively to implement the methods described at 48, 49 and 54. However, in order to achieve the most intuitive and MS Windows like interface, and because effective two-way communication between JavaScript and JAVA had been achieved (See FIG. 9), the languages proposed appear to best support the particular user input technique.

For example, FIG. 23 shows an actual file selection window 2300, implemented by the invention. This type of file selection window is available in JavaScript/HTML, but not supported by JAVA for applets. File selection window 2300 greatly enhances the interface for the user, as the image, sound clip, or video clip names need not be memorized. File selection window 2300 further eliminates possible operator error when typing in a pathname or filename. The present invention utilized the strengths of JavaScript/HTML with the power of JAVA to create a unique browser based interface solution. In one implementation, the HTML form element "INPUT type=file" was embedded in a JavaScript pop-up window to create the file selection window. The file selection window returns a string value of the image (or other file type) pathname to the pop-up window. The pop-up window's JavaScript then could be used to call a JavaScript function in the panel (panel 400) which:

- 1: Reads the pathname value in the pop-up window.
- 2: Creates a string version of a valid URL by adding the correct URL protocol to the string.
- 3: Updates the panel's database (interface's database 360).
- 4: Calls a JAVA method in the build engine, which casts the string value of the URL into a URL object, creates an image object which is then drawn on the screen, and updates its internal database.

User inputs that are a selection from a JAVA window object (48) permit the implementation of a vast array of intelligent user input interface objects, from sliders to dials, which are extremely intuitive and significantly enhance the user's ergonomic experience. In one implementation, user input interface objects are supported as follows. When a selection from a JAVA window is detected, a pop-up window (applet) is presented (associated with the feature being manipulated, e.g., color, volume) and an engine method is called to begin two-way communication (for passing as arguments any necessary status information). The engine begins polling a JAVA abstract object waiting for a static variable's value to change. The pop-up applet processes the value as defined by a user interaction event, and updates the static variable in that same JAVA abstract object with the new value. Upon detecting a change in the polled static variable, the engine calls the necessary methods to process that new value. These methods include can include a brightness filter that is applied to the image bitmap utilizing techniques very similar to that of that employed by the "fade in" and "fade out" animations, described in association with FIG. 33

User inputs for a selection from a JavaScript pop-up window (49) can be made in a manner identical to that of making a selection from a dialog box under MS Windows, including the use of tabbed JavaScript pop-up windows. In one implementation when a selection from a JavaScript pop-up window is detected, the panel's (panel 400) JavaScript opens a pop-up window. The pop-up window's initial values are set from a JavaScript database defined in the panel or by the panel calling the engine for the current values and then setting the initial values. In a tabbed JavaScript window, clicking on a tab will call the pop-up window's JavaScript in order to change the state and appearance of the tabbed JavaScript window in the expected way. The pop-up window's JavaScript calls the panel's JavaScript when a completion event occurs. The panel's

12

JavaScript reads or the pop-up window's JavaScript writes the pop-up window's field values, causing the panel's database to be updated, and the panel then calls the appropriate build engine 352 method, passing as arguments the necessary data and status conditions. Initializing the pop-up window's values and updating the panel's database upon completion can alternatively be implemented by JavaScript functions executed within the pop-up window's HTML file.

In addition, there are interface extensions that can extend beyond the usual MS Windows implementations. One is support for a selection from a dual spin control at 50A. Screen shots FIGS. 42-45 show a visualization of an implementation of this interface technique. Screen shot FIG. 42 shows the mouse placed over an upper spin control. Screen shot FIG. 43 shows the result after the user clicked once on the upper spin control. Notice that the value has been incremented by 1, and the text button object is now at a larger point size. Screen shot FIG. 44 shows a combo box list selected by the mouse with the user about to select a significantly larger point size. Screen shot FIG. 45 shows the result of that selection, including the effect on the text button object.

In one implementation, dual spin controls are supported as follows. Each spin control has three visual states, so that when the user places the mouse over the control it appears to light up, and when the mouse button is depressed (pressed down), the spin control is modified to give the appearance of being pressed. JavaScript methods are called in the panel (panel 400) to:

- 1: process each mouse click event over either spin control,
- 2: range check as necessary,
- 3: update the value in the HTML frame object residing in the pop-up window,
- 4: update the JavaScript (panel 400) database,
- 5: call the build engine 352, if necessary, passing the necessary value and status.

If the mouse is clicked on a combo box, the selection window opens in the usual way. If a mouse click in that window is detected, another JavaScript method in the panel 400 is called to update the JavaScript database, and call the build engine 352, if necessary, passing the necessary value and status as function call arguments.

Another interface extension is selection from a JavaScript child window at 50B. This technique helps simplify the number of choices given to the user in a complex pop-up window operation. A selection from a JavaScript child window can be supported as follows. The panel's (panel 400) JavaScript opens the pop-up window. The pop-up window and its child pop-up windows' initial values are set from the JavaScript database defined in the panel 400. The pop-up window's JavaScript opens the child pop-up window and sets its initial values. The child pop-up window's JavaScript calls the pop-up window's JavaScript when a completion event occurs. The pop-up window's JavaScript reads the child pop-up window's values, sets those values to its own internally defined variables, and calls the panel's JavaScript. The panel's JavaScript reads the pop-up window's values (which include the settings for its own fields as well as those of its child windows), updates its database, and calls the appropriate build engine 352 method, passing as arguments the necessary data and status conditions. Screen shots FIGS. 46-47 show a visualization of an implementation of a JavaScript child window. Screen shot FIG. 46 show a change text button style pop-up window. Screen shot FIG. 47 shows the result after the user selected the "Define the Mouse Down Text Button Style" child pop-up window.



US 7,594,168 B2

13

Direct text entry is supported at any arbitrary cursor location. In one implementation, "text areas" are utilized in an unconventional way, in order to support full text entry, text editing, text button and paragraph styles, and reformat. Direct text entry can be defined at any arbitrary cursor location, and then text can be dragged to any other arbitrary location.

Text areas are objects that are utilized by JAVA primarily as an interface object for the implementation of a form and are generally "added" to the screen at the initialization time of a JAVA applet. Text areas are decidedly not WYSIWYG. The present invention creates text areas dynamically. Screen shots FIG. 38 through FIG. 41 show a visualization for an implementation of this technique. Screen shot FIG. 38 shows the user selecting a text object from the create text icon object from a tool bar of the panel (panel 400). When the text icon object is selected, the cursor shape is changed to indicate the selection while the text icon object is in the select state. Screen Shot FIG. 39 shows that the cursor has changed shape and that the user has placed the cursor at an arbitrary location on the web page. Screen shot FIG. 40 shows the result after the user has clicked the mouse. A text insertion point and a selection rectangle are drawn at the arbitrary web page location. Screen shot FIG. 41 shows the result after the user has pressed the letter "W" on the keyboard. As can be seen in screen shot FIG. 41, a draw method associated with the build process immediately hides the text area. However, text editor methods associated with the build process continue to utilize the text area as a hidden, dynamically resizing frame, whose size is subject to text button or paragraph style settings, by the amount of text, by the text's orientation (vertical or horizontal) and by the text's font style(s) and font size(s). As the build engine 352 detects a relevant mouse event or keyboard event, the build engine 352 updates the necessary variables that are defined as return values in specified build engine methods. Polling technology (see FIG. 9) retrieves the relevant values and calls the necessary JavaScript method for processing. In one implementation, these same techniques (text area techniques) are used in the scaling technology (See FIG. 30). Since the direct text entry and editing processes bypass completely the interface and the JavaScript code, the polling technology (See FIG. 9) is used to pass the text string values back to the JavaScript database, in order for the interface's pop-up windows to be correctly initialized for subsequent text operations.

Direct text processing at 51 begins with the build engine 352 detecting a "Mouse Drag" or a "Mouse Double Click" event. In one implementation of the present invention, if a mouse drag event is detected, the entire initial anchor word (assuming the "mouse down" event placed the text insertion point within a word) is selected as well as the entire closing anchor word. If a double click event occurs over a word, the entire word is selected. If a double click event is detected over a special hot zone (for example, just to the left of a paragraph line), then an integral number of words are selected. Appropriate four-dimensional variables are set, and a draw system is called. The draw system paints the selected line segment in the marked text background and text color. The build engine 352 then sets a return flag to be read by the polling technology (FIG. 9). A panel JavaScript poller (FIG. 9) detects this flag and redraws the panel's "Text" menu object showing the choices available when text is selected. In one implementation, the "Text" menu includes choices of "Text Style", "Hot Link", "Preferences", and "Format". The states for the tool bar icon objects of "Bold", "Italic" and "Underline" are set appropriately as is the setting for the point size interactive drop-down list. The panel's JavaScript then calls an appropriate build engine method that resets the flag. If the panel's

14

JavaScript detects the user selecting the "Text Style", "Hot Link", "Preferences" or "Format" choices, it creates the appropriate pop-up window. Upon detecting a user completion event, the panel's JavaScript reads the data settings in the pop-up window, closes that pop-up window, and sends this data to an appropriate build engine method for processing (See FIG. 11).

Dynamic web page resizing at 52 is invoked when the build engine 352 detects a user initiated web page resize event. This could be caused by the "Open" or "Web Site" commands from the "File" menu, or from a "Zoom" command from the "View" menu. This technology is explained in detail below in association with FIG. 22.

Direct object manipulation at 53 includes dragging of any object, resizing of non-text objects, rotation and other image manipulation functions, as required. The processing for direct object manipulation begins by analyzing the type of object selected and the state of the object, as set by the interface based on a user's panel selection. The build engine 352 then changes the mouse cursor's appearance, and the type of selection rectangle, including which attachment points, if any, should be drawn and activated. (See FIG. 10 for the mouse event processing technology and FIG. 12 for image processing technology). In one implementation, the same direct object manipulation polling technology is used as described above with regard to direct text entry.

If a selection of an interactive field, interactive drop-down list object, or a toll bar icon object from the JavaScript panel is detected at 54, then the following steps can be invoked, depending on the selection. The point size of a paragraph, a marked text range inside a paragraph or text button object can be changed. The state of an object's 3D frame can be changed. In one implementation, three states for an object frame are supported. The 3D frame can be drawn as a "raised" 3D object, as a "depressed" 3D object, or as a "raised" 3D object that turns into a "depressed" 3D object if a mouse down event is detected over the object to which the 3D frame is assigned. An object's style can be changed. The current web page can be changed. Finally, any other operation that has been defined by a tool bar icon object in the panel can be invoked. This includes the "file" menu choices of new, open and save, the "edit" menu choices of cut, copy and paste, inserting a text, button or image object onto the web page, applying or removing the bold, italic, and underline text attributes for a text or button object, centering or uncentering any web page object, setting the animation for a button or image object, changing the zoom level of the web page, or previewing the web site.

As each new user input is received and processed in accordance with the steps shown in FIG. 6, at all times the internal databases of the JavaScript panel and the build engine 352 are maintained completely in synchronization. Synchronization is maintained so that: all status information displayed by the panel is current and correct; all data and status information passed to the build engine 352 from the interface are consistent with the build engine's state at any given time; the values in all pop-up windows are correctly initialized. In order to meet these requirements, all of the variables in the JavaScript panel database are explicitly "typed", to be compliant with the strict variable typing methodology generally imposed in all full featured programming languages such as Java. As JavaScript does not explicitly type anything, where using JavaScript herein, all string, Boolean, and integer variables are typed. Full two-way real time communication support between the JavaScript/HTML interface and the JAVA build engine 352 is provided as described below in association with FIG. 9.

US 7,594,168 B2

15

FIG. 7a shows four tools utilized for an implementation of the pop-up window and panel interface technology (15 of FIG. 3). The panel and pop-up windows make extensive use of JavaScript mouse events, including onMouseDown, onMouseUp, onMouseOver, onMouseOut, onClick and onChange methods (56). The pop-up windows make extensive use of the JavaScript onLoad and onUnload methods. In one implementation, when a pop-up window is loaded by the panel, the panel goes into a wait loop, set for 5 times a second using the JavaScript setTimeout method, interrogating in each loop whether the pop-up window's status flag has been set. Meanwhile the pop-up window, when loaded by the browser, executes the onLoad method in order to set a flag in the panel informing the panel that the pop-up window is now loaded. Upon detecting the load event completion, the panel then proceeds to initialize the fields in the pop-up window. The panel will always close a pop-up window after detecting its completion event. However, if the user has closed the pop-up window in a non-standard way, the pop-up window executes the onUnload JavaScript method, which sets a flag in the panel notifying it that the pop-up window has been closed.

The JavaScript code in the panel and in all pop-up windows make extensive use of JavaScript method onKeyDown for the following operations:

- 1: When the focus is on the icon representing completion ("OK" is used in many MS Windows applications) causing the enter key to initiate a pop-up window/panel completion event.
- 2: When the focus is on the icon representing cancellation ("cancel" is used in many MS Windows applications) causing the Esc key to initiate a pop-up window/panel cancellation event.
- 3: When the focus is on any pop-up window or panel object, such as a data entry field, a check box, a radio button, a drop-down and scrollable list, a scrollable list, an icon, or a DHTML tab object (discussed below), the navigation keys are captured by the onKeyDown method, a JavaScript function is called, and the appropriate change is made.

For all pop-up window and panel objects, when the Tab key or the combination of the Tab key with the Shift key are detected, the onFocus JavaScript method is employed and the focus moves to the appropriate pop-up window object. If the pop-up window or panel object is a data entry field, drop-down list or a scrollable list, all cursor key operations are detected and the insertion point is adjusted accordingly. If the pop-up window or panel object is a check box, radio button a icon, or a DHTML tab object, and a cursor key (up, down, left, right, home and end keys, with or without the Ctrl or Shift keys) is detected, the onFocus JavaScript method is employed and the focus moves to the appropriate pop-up window object.

One methodology for this feature requires that all keyboard events be monitored, at all times. When the scan code for the enter key is detected, the appropriate JavaScript function is called to close a pop-up window and to call the appropriate JavaScript function for processing of the relevant data (updated in the window) and communicating, as necessary, with the build engine 352. In another implementation, rather than the panel going into a wait loop awaiting notification from the pop-up window for data initialization purposes, the pop-up window, when loaded, executes the onLoad JavaScript method, and reads the required data values directly from the panel's database, utilizing the JavaScript "opener.fieldname.value" technique. Similarly, the pop-up window, when detecting its completion event, updates the panel's database

16

with the revised values from its own fields and then calls the appropriate JavaScript function in the panel for further processing. Both implementations, and any combinations, assure that the pop-up windows are correctly initialized, the panel's database is correctly updated, and the data is successfully sent to the build engine 352 for processing.

Extensive use of JavaScript technology is employed to enhance the user interface and for communication between the various HTML frames and/or files, within a given HTML frame or file, between an HTML frame and the JAVA engine, and as a bridge between two different JAVA applets (57). Extensive use is made of JavaScript arrays to store the values of all page and object attributes, to initialize the correct values in all pop-up windows, and to pass data and status to the engine. Various JavaScript techniques are employed to "type" all variables (JavaScript does not explicitly type anything as described above) as a prerequisite for passing values to the build engine 352. Variables that should be typed as strings, integers and Booleans are typed through the use of "Eval" and "New" JavaScript functions. The choice of color, found in most pop-up windows to define one or more color elements, can be implemented utilizing several innovative JavaScript techniques. They include:

- 1: Defining a complex image map through a JavaScript function utilizing arrays. Screen shot FIG. 48 shows a visualization of an image map. A JavaScript computational loop utilizing arrays can be used to define each individual rectangle in this color palette with its appropriate RGB value and a function call to the appropriate JavaScript method.
- 2: Limiting the color choices from the image map to only those colors that are designated as safe colors. Safe Colors are the subset of all colors that are browser independent, assuring a consistent color look across all browsers.
- 3: Supporting a dual color selection technology. The user can be presented with a color palette and can click on a particular color in the color palette. Image map technology can call a JavaScript function, which converts that choice into a RGB numeric definition. This definition updates the RGB values shown in screen shot FIG. 48, as well as passing those values, through an appropriate function call, to a build engine JAVA method. The build engine 352 will then draw the actual color immediately on the web page. Alternatively, the user can select a value from Red, Green or Blue selection lists, which can be implemented using an HTML drop-down list form object. The value selected is then processed by an appropriate JavaScript function call to a build engine method, which converts the RGB to a JAVA compliant value, and then draws the actual color on the web page.
- 4: Supporting True Transparency. For appropriate color elements, such as the background for a text button object, the user can choose, either from the color palette by clicking on a "transparency" rectangle, as described above, or by selecting "TR" from a Red, Green or Blue selection list. This choice is then processed by an appropriate JavaScript function call to a build engine method, that in turn sets a particular flag for the draw system (of the Build Tool) to not draw a background color for that object.

Innovative techniques are used to enable JavaScript to dynamically create HTML code based on real time conditions. Cookies can be used for data communication between HTML frames and HTML files, some of which were created in real time. Many unique combinations of HTML elements, including frames, forms, and tables, enhanced by JavaScript

US 7,594,168 B2

17

code, are utilized to create extensions beyond that of the MS Windows interface (58). For example, a dual combo box/spin control for both small and large numeric incremental jumps can be implemented by a combination of form and table elements, mouse events, and JavaScript methods.

Extensive use of Cascading Style Sheets (CSS) was employed to create a consistent look for all pop-up windows, and for precision placement of various HTML elements (59).

FIG. 7b shows a detailed view of the build time techniques for implementation of panel interface objects, including the menu bar, menus and sub-menus, the tool bars, status fields, interactive fields, and interactive pull down lists, in accordance with one implementation of the present invention (15 of FIG. 3). These techniques create panel interface objects that have the same look and feel of those which are implemented under the various Microsoft Windows Operating Systems. In one implementation of the present invention, the status fields, interactive fields, and interactive drop-down lists are defined as HTML form objects (text boxes and lists) embedded within DHTML objects. The menu bar, menus and sub-menus, and the tool bars can be defined as pure DHTML objects. However, Cascading Style Sheets can be used for all panel interface objects; although more extensively with DHTML objects as will be described below. In an alternative implementation of the present invention, the status fields and interactive drop-down lists are defined as pure DHTML objects.

In one implementation of the present invention the menu bar at 270 is defined as sets of DHTML objects, each set corresponding to a menu command. Each set consists of four DHTML objects with absolute screen positioning, one defining the DHTML object in the Mouse Over state at 278, the second for the Mouse Down state at 279, the third for the Active state, and the fourth for the Inactive state. Each state has a different CSS style assigned, which defines the visual appearance of that state. When the build tool is initialized at FIG. 5, the appropriate menu commands are initialized as active or inactive at 277. If the menu command is defined to be inactive, that DHTML inactive object is assigned by a JavaScript function to the "visible" style attribute, while the other three DHTML objects are assigned the "hidden" style attribute. Screen shot FIG. 38 shows a visualization of the "Interactions" menu command in the inactive state. In the inactive state all user interactions are ignored. If the menu command is defined to be active, that DHTML active object is assigned by a JavaScript function to the "visible" style attribute, while the other three DHTML objects are assigned the "hidden" style attribute. While in the active state, the JavaScript functions for "onMouseDown", "onMouseUp", "onMouseOver" and "onMouseOut" are implemented. If a Mouse Down user interaction event is detected over an active menu DHTML object at 279, a menu command specific JavaScript function is called. This function sets the DHTML object for the Mouse Down state to the "visible" style attribute, calls a generalized JavaScript function to reset the visibility states all the other appropriate DHTML objects, set certain status variables, and set the DHTML object which defines the menu associated with that menu command to the "visible" style attribute. Screen shot FIG. 37 shows a visualization of the "Image" menu command after having received a mouse down event, with its associated menu 420 having been set to the "visible" style attribute. If a mouse up user interaction event is detected over an active menu DHTML object at 281, a generalized JavaScript function is called in which the DHTML object defining the mouse over state is passed as a function call argument. This function sets the DHTML object defining the mouse over state to the "hidden"

18

style attribute thus resulting in the appearance as shown for the image menu command in screen shot FIG. 37, even when the mouse has been moved off the menu object. If a mouse over user interaction event is detected over an active menu DHTML object at 278, a generalized JavaScript function is called in which three DHTML objects are passed as function call arguments as well as a menu command name. These DHTML objects are the ones defining the mouse over state, the mouse down state, and the associated menu. This JavaScript function first tests to see if a menu has been activated by a previous mouse down event and is still visible. If so, a generalized "reset visibility states" function is called, then both the mouse down and associated menu objects are set to visible. Finally the same menu specific function is called as with the mouse down event. If no menu is visible, then the object associated with mouse over state is set to visible. If a mouse off user interaction event is detected over an active menu DHTML object at 281, a generalized JavaScript function is called in which the mouse over DHTML object and the menu command name are sent as arguments. Logic tests are made to determine which menu command object has been left, as well as whether any menus are currently visible. Depending upon the results, the mouse over DHTML object may be set to hidden.

In one implementation of the present invention the menus and sub-menus at 271 are defined as a set of DHTML objects, one for each menu choice, nested inside an DHTML object that defines the entire menu. Each menu object is given absolute positioning, while the menu items are given absolute positioning relative the menu objects origin. Both the entire menu and each choice are assigned CSS styles to define their visual appearances. For each menu choice the JavaScript functions for "onClick", "onMouseOver" and "onMouseOut" are implemented. If a mouse click event is detected at 280 and no sub-menu is defined, a feature specific JavaScript function is called. First the menu bar and the menus are set to their appropriate visibility states. Then setting their visibility attribute style to "visible" activates the appropriate tool bar icon DHTML objects. Finally the feature specific JavaScript code is executed as discussed herewithin, which may cause a pop-up window to be displayed, the Panel's database to be updated, and/or the build engine 352 to be called. If a mouse over event is detected at 278 and no sub-menu is defined, a generalized JavaScript function is called in which the menu choice object is passed as an argument. This function first calls a generalized JavaScript function to close any pop-up windows, then set a status variable and finally executes DHTML commands to set the correct text and background colors for the object. If a mouse off event is detected at 282 and no sub-menu is defined for a menu choice either immediately above or below, a generalized JavaScript function is called in which the menu choice object is passed as an argument. A status variable is set and DHTML commands are executed to set the correct text and background colors for the object. If a sub-menu is defined for a menu choice object, then the same sub-menu specific JavaScript function are called for both mouse click or mouse over events. This function performs the same steps as that of the generalized function that was called for a mouse over event, as well as setting the sub-menu object and its menu choice objects to the visible state. Screen shot FIG. 37 shows a visualization of the menu bar's "Image" command having been activated, the drawing of its associated menu 420, the selection of the "Enhance" menu choice, and the drawing of the "Enhance" sub-menu 430. In the event that the cursor is moved to an adjacent menu choice under the "Image" menu, such as "Animation." or "Rotate", then a specific JavaScript function is called which,

US 7,594,168 B2

19

in addition to the functions executed by the generalized JavaScript mouse over function, also hides the “Enhance” sub-menu.

In one implementation of the present invention, the tool bars at 272 are defined as a DHTML objects, and a set of DHTML objects are defined for a tool icon. The tool is given absolute positioning and is assigned a CSS style in order to define its visual appearance. Each tool icon is assigned a set of three DHTML objects all with absolute screen positioning. The first DHTML object defines the mouse over state at 278, the second for the mouse down state at 279, and the third for the active state. Each state has a different CSS style assigned, which defines the visual appearance of that state. For each tool icon active state object the JavaScript functions for “onClick”, “onMouseDown”, “onMouseUp”, “onMouseOver” and “onMouseOut” are implemented. GIF images are defined for the tool bar DHTML objects, and may be always visible. The inactive “grayed out” representations for each tool icon can be drawn on this image. When the build tool is initialized at FIG. 5, the appropriate tool icon objects are defined as active or inactive at 277. The inactive state for an tool icon is represented when all three of its associated objects are assigned the visibility style of “hidden”. Screen shot FIG. 38 shows a visualization for several inactive tool icons including the icon commands for bold, italic, underline, left and centered. All user interaction events are ignored in the inactive state. If the tool icon, based on the state of the build engine and based on the polling technology described below, is set to an active state, then the tool icon’s active state object is set to the visibility style of “visible”. If a mouse click event is then detected at 280, a feature specific JavaScript function is called in a manner identical to that for a mouse click event over a menu choice object as described above. If mouse down or mouse up events are detected at 279 or 281, then respective generalized JavaScript functions are called, in which the DHTML object defining the mouse down state is passed as a function call argument. If a mouse down event was detected, then the generalized function sets the tool icon’s mouse down object to the “visible” state. If a mouse up event was detected, then the generalized function sets the tool icon’s mouse down object to the “hidden” state. If mouse over or mouse out events are detected at 278 or 282, then respective generalized JavaScript functions are called, in which the DHTML object defining the mouse over state is passed as a function call argument. If a mouse over event was detected, then the generalized function sets the tool icon’s mouse over object to the “visible” state. If a mouse off event was detected, then the generalized function sets the tool icon’s mouse over object to the “hidden” state. Screen shot FIG. 37 shows a visualization of the button tool icon with both its associated mouse down and active objects set to “visible”. Screen shot FIG. 38 shows a visualization of the text tool icon with both its associated mouse over and active objects set to “visible”.

In one implementation of the present invention, the status fields at 273 and the interactive fields at 274 are defined as HTML text boxes. In an alternative implementation status fields are defined as DHTML objects. For both of these panel interface object types, under certain conditions, the panel draws status information into these panel interface objects. This information can result from user input as discussed in FIG. 6, or through the polling and two-way communication technology between the interface and the build engine 352 as discussed below. In one implementation of the present invention the status fields are:

- 1: The color of the selected web page object, in which the red, green and blue settings are shown.

20

- 2: The animation state of the selected button or image object.
- 3: The zoom level for the current web page.
- 4: The point size for the selected text or button object.
- 5: The horizontal position, in pixels, of the mouse cursor.
- 6: The vertical position, in pixels, of the mouse cursor.
- 7: The type of web page object (text, button, image, table, form object, draw object, etc.,) if selected. The type of object that the mouse is over, if no object is selected.
- 8: The width, in pixels, of web page object (text, button, image, table, form object, draw object, etc.,) if selected. The width of the object that the mouse is over, if no object is selected.
- 9: The height, in pixels, of web page object (text, button, image, table, form object, draw object, etc.,) if selected. The height of the object that the mouse is over, if no object is selected.

Screen shot FIG. 38 shows a visualization of the status fields in one implementation of the invention 450. In an alternate implementation using DHTML objects, the status fields will appear two-dimensional rather than the three-dimensional look currently shown.

There is one interactive field defined in one implementation of the present invention. Screen shot FIG. 37 at 460 shows a visualization of the page number interactive field. In addition to the current web page being displayed, either as a number in one implementation or as a user defined name in an alternative implementation, the user can place the cursor into this field and enter the desired page to go to. A click at 280 or Enter Key event will execute this function.

In one implementation of the present invention, the interactive drop-down lists at 275 are defined as HTML form lists. In an alternative implementation, status fields are defined as DHTML objects. For both of these panel interface object types, under certain conditions, the panel draws status information into these panel interface objects. The interactive drop-down lists behave in a manner very similar to that of interactive fields, except that when selected, a selection list drops down for selection. Depending upon the number of entries in the list, an elevator object may be drawn. The operations of selecting the interactive pull down list, the selecting of a list item, or the operation of the elevator is identical to that of comparable MS Windows objects. In one implementation of the present invention the interactive pull down list are:

- 1: Zoom. This interface object has dual spin controls as described above and is always selectable except when in a preview mode. It shows the current zoom level.
- 2: Button Style. This interface object is always selectable except when in preview. It shows the button style of the currently selected button, if any. Changing the button style will change the style of the currently selected button, and/or define the style of the next button to be created.
- 3: Point Size. This interface object has dual spin controls as described above and is selectable when a text or button object is selected. It shows the point size of the currently selected text or button object, if any. Changing the point size will change the point size of the currently selected text or button object.
- 4: Paragraph Style. This interface object is always selectable except when in preview. It shows the paragraph style of the currently selected paragraph, if any. Changing the paragraph style will change the style of the currently selected paragraph, and/or define the style of the next paragraph to be created.

US 7,594,168 B2

21

5: Frame State: The state of the 3D frame (none, raised, pressed or live) of the currently selected text, button, or image object.

6: Image Style. This interface object is always selectable except when in preview. It shows the image style of the currently selected image, if any. Changing the image style will change the style of the currently selected image, and/or define the style of the next image to be created.

Screen shot FIG. 37 shows a visualization of interactive drop-down lists 470. In an alternate implementation using DHTML objects, the interactive drop-down lists will appear two-dimensional rather than the three dimensional look currently shown.

Tool bar icon objects, status fields, interactive fields, and interactive pull down lists all show feedback of the current build engine state. The technology utilized by one implementation of the invention is described below.

FIG. 7c shows a detailed view of the of the build time techniques for implementation of tabbed pop-up windows (15 of FIG. 3). These techniques create a pop-up window interface that visually and behaviorally is identical to that which is implemented as dialog boxes under the various MicroSoft Windows Operating Systems. Pop-up windows can be non-tabbed as described in FIG. 7a, or can have from two to as many as 10 or more tabs, depending upon the complexity of the choices available to the user for a given feature. In one implementation of the present invention each tab at 283 is defined as a DHTML object at 284. The tab is given absolute positioning and is assigned a CSS style at 286 in order to define its visual appearance. When a click is detected through the JavaScript "onClick" function, a tab specific JavaScript function at 285 is called within the pop-up window's HTML file. This function sets the display style attribute for the DHTML objects that define the settings for all the non-selected tabs to the display style attribute of "none". The DHTML objects that define the GIF image of the non-selected tab file representations are also set to the display style attribute of "none". The display style attribute for the DHTML objects that define the settings of the currently selected tab and the GIF image that depicts the selected tab file representation is set to the display style attribute of "". If there is to a change of the focus of the selected field within the now to be visible tab specific choices, the focus attribute for that object is executed. Screen shot FIG. 37 shows a visualization of a tabbed pop-up window, and screen shot FIG. 63 shows a collage of four views of a tabbed pop-up window with four tabs. Notice that each state of the tabbed pop-up window has a different tab file representation, showing the selected tab as being in the forefront.

The interface technology of the invention, in addition to its utilization as part of a web-based web site generation tool, can be used to provide a general purpose interface for all web-based applications that want a MS Windows compliant interface.

A process for updating the internal database of the build engine 352 is shown schematically in FIG. 8. The database is compact and efficient in order to meet the performance requirements for the run time process. The database handles a wide selection of data objects, including multi media objects such as colors, fonts, images, sound clips, URLs, threads, and video. The database supports a multi level animation, transformation, and time line model (discussed in greater detail below). The database complies with the differing rules imposed by the various popular browser security managers.

The process begins by determining the type of data to be updated at 60. Data that defines generic web site settings (See

22

FIG. 21a), screen resolution values (See FIG. 21a and FIG. 24), and the web page high watermark setting (See FIG. 24) can be stored in a header record as boolean and integer variables and URL and color objects at 62 and 63. Data that defines web page, paragraph, text button, and image style and text button, image and paragraph high watermark settings can be stored in one-dimensional arrays as boolean, integer and string variables and URL, font, image or thread objects at 61 and 64. The URL, color, font, image and thread objects can also be created as required at 64.

Data that defines text button, image, paragraph, or other parent objects and paragraph line high watermark settings can be stored in two-dimensional arrays (by web page and by object number) as boolean, integer, string, floating point variables and URLs at 65 and 66. Again, the URL, color, font, image, audio clip, video clip, text area and thread objects can also be created as required at 66. Data that defines a paragraph line and paragraph line segment high watermarks can be stored in three-dimensional arrays (by web page, by paragraph number, and by line number) as Boolean, integer or string variables at 67 and 68. Again, the URL, color or font objects can be created as required at 68. Data that defines a paragraph line segment can be stored into four-dimensional arrays (by web page, by paragraph number, by line number and by line number segment) as Boolean, integer or string variables or URL, color and font objects at 67 and 68.

As a data field is added, changed or deleted, a determination is made at 69 on whether a value for a given high watermark needs to be changed. If so, it is updated. As a specific method in the build engine is called, a determination is made at 70 on whether a feature flag needs to be set. For example, if a particular JAVA method is called, which requires an instance of a certain JAVA Class to be executed by the run time engine, then that JAVA Class is flagged, as well as any supporting methods, variables and/or object definitions. The use of these flags is described in greater detail below in association with FIG. 25 and FIG. 27 to create a compact and efficient customized run time environment.

In one implementation, the header record, the style record, the web page record, and the object records, are carefully defined in a specific order, written in that order, and explicitly cast by object type when read by the run time engine. Exception handling can be implemented to recover from any errors. This helps assure that data integrity is maintained throughout the build and run time processes.

FIG. 9 details the polling process (16 of FIG. 3). The polling technology is essential for creating the necessary two-way real time communication between the JavaScript/HTML interface and the JAVA build engine. Since there is no particular difficulty for JavaScript to be able to call and pass values directly to JAVA methods, the technological challenge is to find a reasonable technique to enable JAVA to communicate back to JavaScript. The polling technology is generic, and workable across all the current browsers. The polling technology is flexible, as there are no real constraints as to what data could be communicated from the build engine to the interface, and this communication can occur at any time. The polling technology is reasonably efficient, so that the performance of the build process is not significantly affected.

In one implementation, two different techniques were utilized to implement this capability. The first was to place the build engine inside a JAVA wrapper. The JAVA wrapper accepts direct communication from JavaScript function calls, interrogates a particular JAVA build engine method, and returns that method's return value back to the calling JavaScript function. The second technique was more unconventional. A polling loop is defined in the panel's (panel 400)

US 7,594,168 B2

23

JavaScript that creates a near continuous, at least from a human perception point of view, dynamic real time link, in order to monitor events occurring inside the build engine. The result is a real time retrieval (from an ergonomic perception point of view) of necessary data and status settings from the build engine back to the interface.

Upon the loading of the panel HTML file, a JavaScript function at 71 (the poller) is immediately called which initiates a polling loop. In one implementation, the polling loop is set at a poll rate of once every 100 milliseconds or less. The polling routine, operating through the JAVA wrapper, calls the build engine in order to read the current horizontal and vertical coordinates of the mouse cursor, and displays them in the panel's status fields (FIG. 37 at 450). The polling routine also polls the build engine in order to detect whether the mouse has moved over a valid object or, by inference, whether a mouse single click, or double click event has occurred. The poller is also constantly requesting the JAVA wrapper to return the status of an error flag in order to inform the user, if necessary, of an unrecoverable error condition, and the reason for it. (See FIG. 10). The poller then calls a panel JavaScript function that, in turn, calls the build engine to reset the flag. The poller constantly requests that the JAVA Wrapper return the status of whether the mouse cursor is over a valid object, and, if so, that object's number, type, height and width. The poller also constantly requests the JAVA wrapper to return the status of whether an object is selected, and, if so, the type and number of that selected object, as well as the objects height, width, and 3D frame state (and the point size of the object's current font if the object is a text button or paragraph object). In addition, if the object is a paragraph, the poller constantly requests the JAVA wrapper to return a flag if a double click or drag mouse event has occurred.

At 72 the polling routine detects a mouse event based on analyzing the return values received.

The poller can infer that the mouse has either moved off or moved on to a valid object at 73 if the mouse over object state has changed or the mouse over object number has changed. If so, the poller updates the relevant interface objects of the panel as appropriate and displays them as necessary, and, depending upon whether the object is a text button object, a paragraph, image object, etc., at 75, begins polling their unique values.

The poller can infer that a single click mouse event has occurred at 74 if the selection state has changed, or the selected object changed. The poller updates the menu bar (FIG. 37 at 410) as appropriate, making the appropriate menu commands either active or inactive. The poller also sets the necessary status variables, and, depending upon whether the newly selected object is a text button object, a text object, image object, etc., at 74, begins polling their unique values. The poller also activates the appropriate menu choice objects inside the "Edit" menu, the "Text" menu, the "Button" menu, the "Image" menu, and the "Interactions" menu objects (FIG. 37 at 420 and 430), depending upon whether an web page object is selected or not, which type of web page object is selected, or, if the selected web page object is a text object, whether text is marked through a drag or double click event. In a similar manner, the poller also sets the values for the interactive field objects (FIG. 37 at 460) and the interactive drop-down list objects (FIG. 37 at 470). More specifically, JavaScript can poll the web page object number. The value of the web page object number can be used to initialize pop-up windows with that object's web page current values, either from the panel's database or, if necessary, by interrogating the build engine's database.

24

The poller can infer that a double click or mouse drag operation has occurred if the flag indicating a double click or mouse drag operation is detected at 75. The poller calls a panel JavaScript function that, in turn, calls the build engine to reset the flag. The poller then calls a panel JavaScript function to display the appropriate panel menu choices. For example, if the double click or mouse drag event occurs within a text object, then the "Text Style and "Hot Link" menu choice objects become active under the panel's "Text" menu object.

Depending on the object type (76), the polling technology performs various functions. If the object is a text object at 77, the values for the paragraph style, point size, object height and width, text color, and the 3D frame status are polled and displayed. The panel's menu objects and the menu choice objects within that are active for a text object are set to the active state, and the non-text menu choice objects are set to the inactive state, which visually means they are unavailable and are "grayed out". In addition, polling can be initiated for the creation of a hot link. If the object is a text button object at 78, the values for the text button style, point size, object height, width, text color, animation state, and 3D frame status are polled and displayed. The menu choice objects inside the panel's menu objects that are active for a text button object are set to the active state, and the non-text button menu choice objects are set to the inactive state, which visually means they are unavailable and are "grayed out". The value of the text button object string is also polled and saved in the panel's database for use when initializing relevant pop-up windows. If the object is an image object at 79, the values for the image style, object height, width, frame color, animation state, and 3D frame status are polled and displayed. Again, the menu choice objects inside the panel's menu objects that are active for a image object are set to the active state and the non-text button menu choice objects are set to the inactive state. In addition, the results of any relevant direct object manipulation are polled and displayed.

FIG. 10 describes a two level error correction technology (17 of FIG. 3) employed by the build process. Initial error checking occurs during the interactions between the user and the interface with the JavaScript error checking code at 80. Any file name, selected by the user through the file selection window or typed in a file pathname (See FIG. 6 at 49) is checked by the panel's JavaScript to assure that it has the correct file type suffix (gif, jpg, au, etc.) at 81.

The panel's JavaScript Code performs range checking at 82 to prevent user error or to prevent the breaking of any internal limits imposed by the build engine. These can include: going to a non-existent web page; exceeding any limit with the dual spin control (i.e. attempting to increment or decrement a point size outside of the legal range, or trying to illegally decrement a value to zero or a minus number; typing in a numeric value that is outside a legal range; and, implicitly creating an object that exceeds a limit imposed by the build engine).

The panel's JavaScript code also checks the file pathname to make sure it contains a valid address, and makes necessary additions or conversions, if necessary, at 83. For example, if the user selected a file from the local disk, the correct URL protocol is appended to the file name in order to make it a valid string representation of a URL address. Any illegal characters for a pathname or a null file pathname entry are also caught at 83. In addition to file pathname validity checking there are other validity checking functions that can be employed by the JavaScript at 83. They include the attempt by the user to enter a non-numeric character into a numeric field, or leaving an essential fill-in field empty.

US 7,594,168 B2

25

The panel's JavaScript then passes these values to the build engine though the arguments of a JAVA method function call at 84. The build engine can utilize the extensive exception handling capability of JAVA at 85 (or that of any other full featured programming language used) to attempt to recover from any processing error. If recovery is not possible, the build engine sets an error flag, utilizing the polling technology (See FIG. 9 at 71). The poller, upon detecting this flag, informs the user, for example, through an alert JavaScript pop-up message, what non-recoverable error has occurred, from which operation, and what actions, if any, the user should take. For example, if the user had selected a corrupted image file, the exception handling technology can inform the user of this fact so that user corrective action can resolve this very common problem. In one implementation, error handling and exception recovery support is provided for a malformed URL, an input or output error, a security manager violation, and a null pointer error.

FIG. 11 shows a process for text entry and text processing (18 of FIG. 3). The process begins when the panel's JavaScript detects the user selecting either "Button" or "Text" icon objects from the panel's tool bar or from their equivalent menu choices under the "Button" or "Text" menus, and calls the appropriate JavaScript function at 86. The JavaScript function, after performing a range check to assure that no internal limits of the build engine are being broken, updates its database, and sets the necessary status variables. The panel's JavaScript then calls the appropriate build engine method, passing the necessary arguments, including the current board number, the internal number to be assigned to the object, the object type, and the current text button or paragraph style at 87. The build engine then updates its internal database and sets the necessary status variables. The build engine also changes the mouse cursor shape to that of a text entry symbol. In one implementation, the mouse cursor is shaped like a crosshair, and can be moved onto the web page (the build frame 402) at an arbitrary location.

The build engine detects a mouse click event through its "mouseDown" method at 88. This method reports to the build engine the exact horizontal and vertical coordinates of the crosshair mouse cursor at the moment the mouse button is pressed. The build engine places these values into its internal database. The polling process is also supported, as discussed in FIG. 9, by placing the necessary return values in the appropriate poll enabled methods.

The build engine creates a dynamically resizable frame utilizing JAVA's "Text Area" object class, whose coordinates and size coincide with that of the draw system for the object as defined below. Other full-featured programming languages, if used by the invention, also possess similar object types. The text area is immediately overdrawn by the draw system's background paint routine. The build engine, utilizing the font metrics as defined by the selected text button or paragraph style, and utilizing the crosshair cursor's coordinates, calls the draw system. The draw system paints the background and then paints an insertion point and a selection rectangle, in the appropriate colors, and with the appropriate height and width, into the appropriate web page location at 89. If the text button or paragraph style has a 3D frame selected, this intelligent ornamental object would also be drawn, in the appropriate color, dimensions, and thickness. Screen shot FIG. 41 shows a visualization of this process. The text insert point is in black, surrounded by a red selection rectangle, and surrounded by a blue 3D frame, as defined by the selected style. The text editor is then initialized by setting the necessary status variables.

26

The build engine waits until a keyboard keystroke is detected. The scan code is interpreted, and if it is a text entry key, the text editor's methods are called at 90. The text editor processes the key event at 91. The build engine employs frame (Text Area) processing methods and draw methods to implement the text entry and text processing functions. As a keyboard key for a text character is pressed, the build engine passes this value to the editor's text entry method, which updates both the text area's frame definition, and the draw system's database. The width of the text area is dynamically resized as necessary. If the object was a paragraph, a check is made on whether a reformat event should occur, based on the paragraph style's definition and the width of the current line's text string. If so, the appropriate text editor reformat method is called, which may cause the text area's vertical dimension to also be resized. A high watermark variable may also be set, for optimization purposes. After the final state of the text area is determined for the text entry keyboard event, the internal database for the text area, and for the paragraph or text button object, are updated. The draw system is called, and the results of the text entry event are drawn on the web page at 94.

In one implementation, the build engine also supports the usual text processing functions found in MS Windows or Macintosh based Word Processors or Desktop Publishers at 92 and 93. For example, if the user single clicks the mouse when over an unselected paragraph or text button object, that object is selected, a selection rectangle is drawn, the mouse cursor shape is changed to a crosshair, and the poller reports the necessary information to the panel's JavaScript. If a mouse click occurs over a selected paragraph or text button object, the editor's "Set Text Insertion Point" method is called. Based on the coordinates of the mouse cursor, and based on a calculation by the build engine as to the nearest line, and the nearest character on that line, the text insertion point can be drawn appropriately, and the necessary status variables are updated. Text entry is then processed as discussed at 91.

If a double click or mouse drag mouse event is detected over a paragraph, an appropriate "text string selection" method is called (See FIG. 6). Based on the coordinates of the mouse cursor, and based on a calculation by the build engine as to what text string should be selected, the internal database is updated, appropriate status variables are set, and the draw system is called for marking the text string at 94. The polling technology is activated as discussed in FIG. 9. The build engine's reformat methods for paragraphs can utilize a "Clean Text Stream" model for calculating line breaks and for updating four-dimensional variables utilized by the draw system in order to draw each paragraph, each paragraph line, and each paragraph line segment in the correct location, with the correct font type, font style, font size, font effect, and background and text string color. Font style refers to a font format such as Normal, Bold, Italic, or Bold Italic. Font effect refers to style overrides such as Underline, Double Underline, Small Caps, Cross Out, Superscript, Subscript, etc. The "Clean Text Stream Model" implemented by the build engine maintains multi-dimensional array pointers and records for every paragraph line and line segment external to the text string defined within the text area. Three-dimensional and four-dimensional variables are updated after each text entry or text editing and processing event in order to assure that the pointers into the paragraph text stream, defined in the text area, are current. The three-dimensional variables that the build engine has implemented can include soft and hard line end pointers for each paragraph line. Their values can be the absolute character positions within the text area text string for that line end. Hard line breaks can be created by the user pressing the enter

US 7,594,168 B2

27

key. Soft line breaks can be created by a reformat method based on a calculation defined below.

The four-dimensional variables can be absolute pointers into the text area text string for the beginning and end of every style override, associated with each paragraph line segment. These style overrides can include hot links, font type, font style, font size, numerous font effects, and text and background colors. For each style override there is an associated style override record that maintains all the font and color settings for that paragraph line segment. Also positional and size data such as start and end pointers into the paragraph text stream, a left offset relative to the paragraph's left origin, a top offset relative to the paragraph's top origin, and the line height. The style override record is created when the build engine detects a mouse drag or mouse double click event within a selected paragraph. When the mouse button is initially pressed, the current paragraph line and current word on that line are calculated in a manner identical to that for calculating the location of the text insertion point on a mouse click operation. The entire word becomes one anchor for the paragraph line segment, while the word defined by the mouse coordinates when the mouse button is released becomes the other anchor. Up to two other paragraph line segments can be implicitly created by the word oriented selection method. If there is text to the left of the first anchor word, and that paragraph line had not previously had a style override defined in it, the text string from the beginning of the paragraph line to the first anchor point has a style override record created for it. The values are set to that of the underlying paragraph.

If style overrides had already been created on that paragraph line, and the anchor word is inside one of them, then that style override's end pointer is adjusted to the start of the anchor word. All other style overrides, if any, to the right of the anchor word are deleted, as overlapping style overrides are not permitted. In a similar manner, the text string, if any, to the right of the last anchor point, up to the line or paragraph end, can also be defined as a style override. If a mouse click occurs before a "text style" operation, then these pointers will be reset. If the panel's JavaScript detects a user selection of "text style" from the "Text" menu, the appropriate pop-up window is drawn and its values initialized from the JavaScript database. Upon detecting a user completion event (i.e., the depressing of the enter key), the panel's JavaScript database is updated and a call is made to an appropriate build engine method, with the necessary data and status information passed as function call arguments. The build engine updates its internal database and calls the reformat method if necessary. The draw system utilizes these four-dimensional variables in order to paint the paragraph line segment style override.

The calculation for the creation or updating of a soft line break begins with the maximum paragraph width, which is set at a percentage of the browser screen width. This percentage is converted to an absolute pixel number based on the web designer's screen resolution. When any text entry or text editing and processing event occurs, a build engine method is called which calculates the width, in pixels, for the current paragraph line, based on the character string in the text area that exists between the previous line end pointer and the current line end pointer. The font definition(s) that are related to this character string are applied, and a string width is calculated. If the string width exceeds that of the maximum paragraph width, an "Overflow" reformat method is called. The overflow reformat method calls a method to determine the settings for the last word on that line, and that word overflows to the following paragraph line. All pointers for the current line, and subsequent lines are updated as necessary, as

28

are all pointers and records to paragraph line segments. If the string width is less than that of the maximum paragraph width, and the text processing operation was not text entry, then an "UnderFlow" Reformat method is called. The underflow reformat method calls a method to determine the width, in pixels, for the first word on the next line. If that word will fit on the current line it is placed there. As before, all pointers for the current line, and subsequent lines are updated as necessary, as are all pointers and records to paragraph line segments. The word oriented selection technique, and the reformat, database, and draw technologies that support it, greatly simplify the text editor and produce a run time engine that is smaller, faster and more reliable.

FIG. 12 shows the operation of the image processing technology utilized by the build engine (19 at FIG. 3). The process begins when the panel's JavaScript detects the user selecting the "Image" icon from the panel's tool bar or the comparable menu choice under the "Image" menu. The appropriate JavaScript function is called at 95, which draws the define image pop-up window. The user then selects an image from the file selection window with the browser, types in the image path-name for the image file on the local disk, or types in the URL for the image that exists on a server. The user could also define a 3D frame for the selected image at this time. Screen shot FIG. 49 shows a visualization of a collage for the define image pop-up window and the user's selection choices under each tab setting. The user can complete the selection process by either pressing the Enter Key or clicking on the "Create Image" icon in the pop-up window. If the Enter Key is pressed, the pop-up window's JavaScript Code utilizes the onKeyDown function, or if a mouse click, the onClick function, as described in FIG. 7, to recognize the completion event. An appropriate error checking JavaScript function is called, which performs a file name error check, a filename validity check, and a range check to assure that no internal limits of the build engine are being broken. If the error checking tests are successful another JavaScript function is called to update the panel's database, and set the necessary status variables.

The panel's JavaScript then calls the appropriate build engine method, passing the necessary arguments, including the current internal web page number, the internal number to be assigned to the image object, the object type, and the current image style at 96. The build engine then updates its internal database and sets the necessary status variables. It also changes the mouse cursor shape to that of an "Image Create" symbol. In one implementation, the mouse cursor is shaped like an arrow. The build engine detects a mouse click event through its "mouseDown" Method at 97. This method reports to the build engine the exact horizontal and vertical coordinates of the arrow mouse cursor at the time the mouse button was pressed, and places these values into its internal database. The polling process is also handled, as discussed in FIG. 9. The build engine then asserts the necessary security permission for reading from the local disk, if necessary, and attempts to create the necessary image object at the current mouse coordinates at 98, while checking for any exception conditions as described in FIG. 10. If no unrecoverable exceptions are reported, the internal database is updated and the draw system is called.

The image processing technology supports direct web page image object interactions at 99, utilizing the communication technology described in FIG. 6. The build engine first processes the mouse event as described in FIG. 7, and places the appropriate values into a poll enabled JAVA method as described in FIG. 9. There are two types of direct web page image object interactions.



US 7,594,168 B2

29

The first occurs by simply selecting the image object with a single mouse click. A red selection rectangle is drawn around the image, as are eight attachment points. When the user has pressed the mouse cursor, the mouse cursor's shape changes to that of an anchor, which is a symbol that can be used when dragging or moving an object. The mouse's location will jump to the origin for the image. In an alternative implementation, the anchor can be defined by the mouse location at the time of the mouse drag operation. In either case, while the mouse is being dragged, the build engine updates its internal database. The build engine also updates its poll-enabled methods for communication with the interface's polling technology at 100. The JavaScript poller reads these values, updates the panel JavaScript database, and updates the panel's interface objects. In a similar way, placing the mouse cursor over an attachment point and dragging will result in an image resizing operation. Screen shots FIG. 50 through FIG. 52 show a visualization of an image dragging operation. Screen shot FIG. 50 shows the cursor over an unselected image. Screen shot FIG. 51 shows the screen state after the left mouse button has been pressed. Notice that the image is now selected and the cursor shape has changed to the drag state. Screen shot FIG. 52 shows the screen state after the mouse has been dragged to the northwest. Notice that the image stayed selected and moved with the mouse. Screen shots FIG. 53 and FIG. 54 show a visualization of an image resizing operation for a normal image. Notice that all eight-attachment points are drawn and active for the selected image. Screen shot FIG. 53 shows the cursor over the upper left attachment point. Notice that the cursor shape has changed to a northwest to southeast resize cursor shape. Screen shot FIG. 54 shows the result after the left mouse button has been pressed over the upper left attachment point and dragged to the northwest. Notice that the image's upper left corner is still under the cursor, the image has resized, and the cursor shape remained unchanged. For image resizing operations with the mouse over and mouse down objects, only the east, southeast, and south attachment points are drawn and active.

The second type of direct web page image object interaction occurs when the panel's JavaScript code detects that the user has selected an image object interaction feature from the panel's "Image" menu. The appropriate JavaScript function is called, which sets the necessary status variables, and then calls the appropriate JAVA method, passing the necessary values as arguments. The JAVA method then sets its necessary status variables, changes the mouse cursor shape as appropriate, depending upon the type of direct image operation, and awaits a direct mouse operation on the image object. Image rotation is an example of this type of direct image interaction. In one implementation, direct image object rotation is realized by utilizing the image rotation technology described in association with FIG. 33 below. Screen shots FIG. 55 and FIG. 56 show a visualization of an image rotation for a normal image. Screen shot FIG. 55 shows the user selecting the rotate command from the "Image" menu. Immediately the cursor's shape changes to the rotate (a dual left/right arrow) cursor, and the selected image's attachment points disappear. Placing the cursor on the image and dragging will cause the image to rotate on an east/west and/or north south axis. Screen shot FIG. 56 shows the result after the mouse was dragged on an east/west plane.

Image object interactions are invoked by selecting from the JavaScript panel, selecting from a JavaScript pop-up window, and by selecting from a JAVA window object at 101, as described in FIG. 6. The initial values in the pop-up window are set from JavaScript's database. After any user interaction, JavaScript's database is updated and the appropriate method

30

in the build engine is called with the necessary settings. The build engine, after updating its internal database, calls the appropriate image processing method. The image processing routine then calls the required image filter(s), which then perform the necessary processing on the image bitmap at 102.

An image filter is a body of code, usually consisting of one or more digital image processing algorithms, which operate on an image bitmap, and create a transformed image bitmap. An image observer can be invoked by the image filter, which then reports when the image bitmap processing has been completed. An image observer is an independent process that monitor's a particular image processing event, such as the execution of an image filter or the reading in of an image file, and reports the status of that process when queried. When the image observer reports a successful completion, the image filter can call the build engine's draw system to display the transformed image bitmap. This interaction between the build engine's image processing method, the image filter(s), the image observer, and the draw system can occur many times, depending upon the image processing operation chosen. Image animations and image transformations, which are technologies that rely heavily on image filters, and the image observer are discussed in greater detail below in association with FIG. 16 and FIG. 17.

FIG. 13 shows a process for implementing text button, image and paragraph style settings (20 of FIG. 3). The initial values for all the settings inside a parent pop-up window and associated child pop-up windows, for a particular style, can be set from the JavaScript database at 103. The settings can include: image object styles, text button object styles and paragraph object styles.

The following settings can be initialized and changed for image object styles.

- a) The following settings are initialized for all image object states (Normal, mouse Over, mouse Down) and can be changed:
  - (1) resize factor.
  - (2) rotation factor.
  - (3) main animation type, speed, number of animation steps (resolution) and number of cycles.
  - (4) image processing factors. (brightness, contrast, etc.)
  - (5) 3d effects and their color values.
  - (6) web page centering attribute.
  - (7) web page scaling attribute.
- b) The following actions are initialized and can be changed.
  - (1) sound effects and audio channels.
  - (2) video files and video channels
  - (3) text button and image pop ups and their attributes (See 1.a above and 2.a below.)
  - (4) click events.
- c) The following transformation settings are initialized and can be changed.
  - (1) the initial delay
  - (2) up to three transformations can be defined with the following settings:
    - (a) which image states should the transformation be from and into.
    - (b) the speed of the transformation.
    - (c) any delay before the next transformation.
  - (3) whether the transformation(s) should occur simultaneously with the enter and exit time line animation or after the enter and before the exit animations.
- d) The following time line settings are initialized and can be changed.
  - (1) the initial delay before the image object's appearance.

US 7,594,168 B2

31

- (2) the enter animation type, speed, and animation resolution.
- (3) the delay after the enter animation and the main animation.
- (4) the exit animation type, speed, and animation resolution.
- (5) the initial delay, after the entrance of the parent object, before the child text button and image object's appearance(s).
- (6) the child object(s) enter animation type, speed, and animation resolution.
- (7) the delay after the child object(s) enter animation.
- (8) the child object(s) exit animation type, speed, and animation resolution.

The following settings can be initialized and changed for text button object styles.

- e) The following attributes are initialized for all text button object states (normal, mouse over, mouse down) and can be changed:
  - (1) all font specifications.
  - (2) vertical state.
  - (3) all color specifications.
  - (4) 3d effects and their color values.
  - (5) web page centering attribute.
  - (6) font processing attributes (available in java 2)
  - (7) scale, shear, and rotate (available in java 2)
- f) The following actions are initialized and can be changed.
  - (1) sound effects and audio channels.
  - (2) video files and video channels
  - (3) text button and image pop ups
  - (4) click events.
- g) The following transformation settings are initialized and can be changed.
  - (1) the initial delay
  - (2) up to three transformations can be defined with the following settings:
    - a) which image states should the transformation be from and into.
    - b) the delay before the next transformation.
  - (3) whether the transformation(s) should occur simultaneously with the enter and exit time line animation or after the enter and before the exit animations.
- h) The time line settings are the same as those defined for image objects. They also are initialized and can be changed.

The following settings can be initialized and changed for paragraph styles. The following attributes are and can be changed:

- i) all font specifications.
- j) all color specifications.
- k) 3d effects and their color values.
- l) web page centering attribute.
- m) the look of hot links, including the text and background colors when the link is active and when the mouse is over the link.

The reference to JAVA 2 under text button object styles refer to the most recent version of JAVA released by Sun Microsystems. This version supports a far more robust two-dimensional processing capability than JAVA 1.6, including significant font processing capabilities and the scaling, shearing, and rotation of objects. Currently, most conventional browsers only support JAVA 1.6. Provisions are made in the invention so that as the then popular browsers support more robust versions of programming languages, those new capabilities can be employed to further enhance the capability of the invention.

32

Referring again to FIG. 13, upon detecting the completion of editing an image, text button or paragraph style, the panel's JavaScript calls a build engine method and passes the required values. The build engine updates its internal database and sets any necessary feature flags at 104.

When an image, text button or paragraph object is created, all the style settings for the currently selected style are applied by the build engine as part of the definition for the newly created object at 105.

If a style is changed, all objects on all internal web pages that are utilizing that style are candidates for being changed to those new values at 106. Flags are kept for every possible style setting for each object. If a given object is edited through the text button, image, or interaction menus or other interface objects of the panel 400, the flags are set for any setting that are changed. If that style is subsequently changed, only those settings that have not had their flags set will be changed for any given object.

FIG. 14 describes the video and audio file and video and audio channel processing techniques employed by the build engine (21 of FIG. 3). A user can select a video or audio special effect (i.e. user input is provided at 107 that indicates a video or audio special effect). The method for activating a video file or video channel is defined in the text button and image object "mouse over" interactive pop-up windows described later at FIG. 16. Methods for defining a video object as a pop-up, or a frozen object, are described with reference to the text button and image object "mouse down" interactive pop-up window also described at FIG. 16. Audio files and audio channels can be defined in both the "mouse over" and "mouse down" interactive pop-up windows also described at FIG. 16. The pop-up or a frozen object settings for audio are also set in the object "mouse down" interactive pop-up windows discussed therein.

As before, the panel JavaScript code initializes any pop-up windows (where the initial values are set from the JavaScript database), captures a file or channel name (from the user input) and performs file and validity error checking upon detecting a user completion action at 108. The build engine is then called, receiving the necessary data and status as function call arguments.

The build engine determines if the audio and video definition is a file pathname or the URL of a live channel at 109, and thereafter initiates its exception handling. If the video or audio definition is a file, the build engine performs the relevant file exception handling checks, and asserts the necessary security permissions. If there were no errors, or the exception handling error was recoverable, the build engine reads and links the video/audio file to the database, and plays the file for user verification at 110. If the video or audio definition was a channel, the necessary pointers are updated in the database, and methods are assigned for efficient transmission, at run time by the run time engine, at 111. The ability of the run time engine to play multiple synchronized audio and video files and channels simultaneously will be described at FIGS. 31-35.

FIG. 15 describes the frames, tables, forms and draw objects technologies employed by the build engine (22 of FIG. 3) in one implementation of the invention. When the panel JavaScript code detects a user action to create a "frame", "table", "form" or "draw object" from an appropriate panel interface object, it draws and initializes the appropriate pop-up window at 112. Upon detecting a user completion action by the pop-up window's JavaScript code, a panel JavaScript function is called to perform the necessary error checking and updating of the panel's database. Panel JavaS-

## US 7,594,168 B2

33

cript thereafter calls the appropriate build engine method(s) passing the necessary data and status values as function call arguments at 113.

The build engine updates its internal database, sets the necessary status values, and initializes, as necessary, appropriate methods for run time processing. In one implementation, the build engine includes definitions to map a given object into a relational database. Also available are a full array of database operations. Support for popular databases (such as Oracle, Informix, Sybase and DB2) are available on a real time interactive basis.

The run generation technologies, as described later in FIGS. 24-27, are also implemented for a given frames, table, form and draw object at 114. The run time technologies, as described later in FIGS. 28-36, are also implemented for a given frame, table, form and draw object at 115.

FIG. 16 describes the user interaction settings and technology employed by the build engine (24 of FIG. 3). Depending upon the type of object currently selected at 116 (if no object is selected no user interaction choices will be available) the panel JavaScript Code draws an appropriate pop-up window. If the selected object was a text button object at 117, or an image object at 119, both "mouse over" and "mouse down" choices will be available from the panel's "Interactions" menu. If the selected object is a paragraph, user interaction definitions can be activated by a double click or a mouse drag event being detected by the build engine at 118.

More specifically, appropriate values are set in a poll-enabled JAVA routine. The JavaScript poller reads the values, and draws the appropriate panel menu choices. The "Text Style", "Hot Link", "Preferences" and "Format" pop-up windows can be chosen. If the hot link choice under the panel's "Text" menu is selected and executed, the hot link definition for internal or external web pages is captured by an appropriate JavaScript function and file pathname error and validity checking is performed. If either the "Text Style", "Hot Link", "Preferences" or "Format" choices under the panel's "Text" menu are selected, the panel's JavaScript draws the appropriate pop-up window. Upon detecting a user completion event, the panel's JavaScript reads the values in the pop-up window and passes the font specification parameters to an appropriate build engine method as function call parameters. The build engine then processes this data, calls a reformat method, updates its internal database, and sets the necessary four-dimensional variables for communication with the draw system.

The normal and "mouse over" foreground and background colors for the hot link, which were defined in a link look pop-up window (available under the "Text" menu of the panel), are utilized by the build engine to draw the hot link. The build engine performs the necessary exception handling, and then updates its internal database.

Based on the panel's JavaScript Code detecting whether the user chose the "mouse over" or "mouse down" choice under the "Interactions" menu) at 120, as well as based on whether an image or text button object is currently selected, the panel's JavaScript code draws the appropriate pop-up window. Initial values for the pop-up windows are set from the panel's database at 121 and 122.

In one implementation, the following user interaction's for the "mouse over" and "mouse down" states for text button and image objects are supported:

- 1: 3D Frame, in a specified color, and selected for a specified 3D appearance, can be defined for text button and image object's "mouse over" and "mouse down" states, as well as for their text, image and video pop-ups.

34

2: The font typeface, font style, font size, font effect(s), text color, and text background color can be defined for a text button object's "mouse over" and "mouse down" states, as well as for the text pop-ups associated from both text button and image objects.

3: Text, image, and video pop-ups can be defined for the text button and image object's "mouse over" state.

4: A sound track (file) can be defined for the text button and image object's "mouse over" state with the following choices:

- a. play once when a "mouse over" event occurs.
- b. play until a click event while on the object.
- c. play until the mouse moves off the object.

5: A sound track (file) can be defined for the text button and image object's "mouse down" state with the following choices:

- a. play once when a mouse click event occurs when over the object.
- b. play until a second click event while on the object.
- c. play until the mouse moves off the object.

6: Both video and sounds can be defined as channels as well as files.

7: The text, image, and video pop-ups can be frozen (i.e. not disappear when the mouse moves off the object after a mouse click event, for both text button and image objects).

8: Text button and image objects can have one of the following click events defined:

- a. go to a specific internal web page.
- b. go to the next internal web page.
- c. return to the parent (calling) web page.
- d. go to an external web page. That web page will replace the current web page.
- e. go to an external web page. That web page will be launched into a new window so that both web pages will be visible and accessible.

After a user completion action is detected, the panel JavaScript code performs the necessary file error and validity checking, updates its database and sets necessary status values, and then calls the appropriate build engine method, passing the necessary data values and status as function call arguments at 123. The build engine updates its internal database, sets the necessary status variables, then draws the appropriate "mouse over" or mouse down" text button or image object states. The build engine also plays the sound or video file for user verification. The run time technology behind the user interactions will be described in greater detail in association with FIG. 36.

FIG. 17 describes the image and text button object animation settings and technology employed by the build engine (25 of FIG. 3). The panel's JavaScript code determines which type of object, and which object number, from the currently selected object, as reported by the poller at 124. When the panel's JavaScript detects a user selection of "Define Image" or "Animate" from the panel's "Image" menu, or a user selection of "Define Button" or "Animate" from the panel's "Button" menu, it draws the appropriate pop-up window and initializes the pop-up window's values from its database at 125 and 126. Screen shot FIG. 57 shows a visualization of one implementation of the "Text Button Animation Specifications" pop-up window and the animation settings available to the user. Screen shot FIG. 58 shows a visualization of one implementation of the "image animation specifications" pop-up window and the animation settings available to the user.

When a user completion event is detected, the panel's JavaScript code captures the values from the pop-up window for the animation type, speed, resolution, and number of

## US 7,594,168 B2

35

animation cycles at 125 and 126, respectively, and updates its database at 127. The panel's JavaScript then calls the appropriate build engine method, passing the necessary data and status values as function call arguments. The build engine updates its internal database and sets the necessary feature flags (See FIG. 8.) Linkage to the appropriate animation method(s) is also set.

A thread object (a thread is an independent asynchronous program that is multiprogrammed with other threads, are defined and executed by the invention, by a JAVA Virtual Machine and by the browser) is created and executed for user verification at 128. Values are set to integrate the given animation thread with the object time line technology (See FIG. 19). Values are set at 129 so that when the thread object is invoked by the run time engine, the appropriate image filter(s) and animation methods are called. The run time technology behind image and text button object animations is described in greater detail in association with FIG. 31 through FIG. 35.

FIG. 18 describes the transformation settings and technology utilized by the build engine (26 of FIG. 3). A transformation is defined as the changing of an object from one state to another based on a timer control, subject to user settings. In one implementation, the available states for text button and image objects are their "normal", "mouse over", "mouse down" and "pop-up" definitions. For text button objects, a transformation is implemented as the instantaneous drawing of one object state while erasing the previous object state. For images, a transformation is the gradual fading out of the previous object state, while, simultaneously, fading into the next object state.

Prior to any user menu selection, the panel's JavaScript code already knows the status of any selected object through the poller mechanism (124 of FIG. 17). This includes what type of object and the object's internal identifying number. When the panel's JavaScript detects a user selection of "Transform" from the panel's "Interactions" menu, it draws an appropriate pop-up window and initializes the pop-up window's values from its database at 130. Screen shot FIG. 59 shows a visualization of one implementation of a "define the transformation for the text button object" pop-up window and the transformation settings available to the user. Screen shot FIG. 60 shows a visualization of one implementation of a "define the transformation for the image object" pop-up window and the transformation settings available to the user. When a user completion event is detected, the panel's JavaScript Code captures the values from the pop-up window based on the object type.

In one implementation, the following settings for text button objects can be specified:

1. The initial delay.
2. Up to three transformations can be defined with the following settings:
  - a. Which image states should the transformation be from and into.
  - b. The delay before the next transformation.
3. Whether the transformation(s) should occur simultaneously with the enter and exit time line animation or after the enter and before the exit animations.

In one implementation, the following settings for image objects can be specified:

1. The initial delay.
2. Up to three transformations can be defined with the following settings:
  - a. Which image states should the transformation be from and into.
  - b. The speed of the transformation.
  - c. The resolution of the transformation.

36

d. Any delay before the next transformation.

3. Whether the transformation(s) should occur simultaneously with the enter and exit time line animation or after the enter and before the exit animations.

The panel's JavaScript updates its database at 131. The panel's JavaScript then calls the appropriate build engine method, passing the necessary data and status values as function call arguments. The build engine updates its internal database and sets the necessary feature flags (See FIG. 8.) Linkage to the appropriate transformation method(s) is also set.

A thread object is created and executed for user verification at 132. Values are set to integrate this transformation thread with the object time line technology (See FIG. 19). Values are set at 133 so that when the run time engine invokes the thread object, the appropriate image filter(s) and transformation methods are called. The run time technology behind image and text button object transformations is described in greater detail below in association with FIG. 31 through FIG. 35.

FIG. 19 describes the text button and image time lines and technology utilized by the build engine (27 of FIG. 3). A time line is an independent asynchronous process that defines the existence of a given text button or image object. An object's time line begins at the time a given web page makes its appearance, either through an immediate draw or through a transition animation. In one implementation, an object time line can be created as an instance of a class, which has a threadable interface. This instance has its own data structures, which define the animations, and transitions associated with the time line definition. An image or text button object time line can spawn child time lines, at a designated moment. A complete description of time line technology, and how they integrate the animation and transformation technologies, will be described below in association with FIG. 31 through FIG. 35.

The build process begins the time line definition process by having the panel's JavaScript determine what is the currently selected object, utilizing the polling technology at 134.

That is, values for the object's appearance time, animation type, speed and resolution are captured. When the panel's JavaScript detects a user selection of "time line" from the panel's "Interactions" menu, it draws the appropriate pop-up window and initializes the pop-up window's values from its database. Screen shot FIG. 61 shows a visualization of a collage of one implementation of a "define the time line for the text button object" tabbed pop-up window and the time line settings available to the user under each tab. Screen shot FIG. 62 shows a visualization of a collage of one implementation of a "define the time line for the image object" pop-up window and the time line settings available to the user under each tab'. When a user completion event is detected, the panel's JavaScript captures the values from the pop-up window based on the object type. The currently available settings, for both text button and image objects, are:

- 1: The initial delay before the image object's appearance.
- 2: The enter animation type, speed, and animation resolution.
- 3: The delay after the enter animation and the main animation.
- 4: The exit animation type, speed, and animation resolution.
- 5: The initial delay, after the entrance of the parent object, before the child text button and image object's appearance(s).
- 6: The child object(s) enter animation type, speed, and animation resolution.
- 7: The delay after the child object(s) enter animation.

US 7,594,168 B2

37

8: The child object(s) exit animation type, speed, and animation resolution.

The panel's JavaScript updates its database at 135. The panel's JavaScript then calls the appropriate build engine method, passing the necessary data and status values as function call arguments. The build engine updates its internal database and sets the necessary feature flags (See FIG. 8.) A build engine method then processes all the data related to this object. The object's animation settings, if any, are integrated into the timeline at 136. The object's transformation settings, if any, are also integrated into the timeline. If an image object, any transformation animation may be executed simultaneously with the appearance and/or exit animations, depending upon the settings. Finally, a multi-level object thread definition is created and executed for user verification. Values are set at 137 so that when the run time engine invokes the thread object, the appropriate image filter(s), animation methods, and transformation methods are called.

FIG. 20 describes the web page transition animations, time line settings and technology utilized by the build engine (28 of FIG. 3). When the panel's JavaScript detects a user selection of "Define" from the panel's "Webpage" menu, it draws the appropriate pop-up window and initializes the pop-up window's values for the current web page from its database at 138. Screen shot FIG. 63 shows a visualization of one implementation of the "define the current web page settings" pop-up window and the web page settings available to the user. In the implementation shown, the choices supported include:

- 1: The web page delay time (which is the delay, after the completion of the last object time line, to the loading of the next web Page).
- 2: The transition animation, which can include a random animation choice. This is the animation applied to the web page when it is loaded and to the previous web page as it departs.
- 3: The number of animation frames per second, which effectively is the resolution of the transition animation.
- 4: The number of animation frames, which effectively defines the time expected for the transition animation to complete.
- 5: The web page's background color. This setting will override the generic setting for the web site, defined in FIG. 21a.
- 6: A web page boarder. This boarder, if selected, will also override the setting for the web site, defined in FIG. 21a. The boarder can be drawn with a 3D effect, taking the background color, and applying a transformation so that, to the human eye, a lighter and darker shade of that color will be drawn appropriately to create a 3D effect.
- 7: The web page's background pattern. This setting will override the generic setting for the web site, defined in FIG. 21a.

The panel's JavaScript updates its database at 139. The panel's JavaScript code then calls the appropriate build engine method, passing the necessary data and status values as function call arguments. The build engine updates its internal database and sets the necessary-feature flags (See FIG. 8.). The web page time line is synchronized with its object time lines by an appropriate build engine method at 140. The web page's appearance delay and transition settings are integrated into the web page time line. Thereafter, a single-level object thread definition is created. Values are set at 141 so that when the thread object is invoked by the run time engine, the appropriate animation methods and object time line threads are called. Again, the run time technology behind web page

38

transition animations and web page time lines is described in greater detail below in association with FIG. 31 through FIG. 35.

FIG. 21a describes the file operations supported by the build engine (29a of FIG. 3). In one implementation, the file operations supported include:

- 1: "Save" at 142 or "Save As" at 143. If the selection from the panel's "File" menu is to "Save" as a web page, the current browser screen height percentage value is sent to the build engine. The build engine updates its internal database and the build process is completed. Thereafter, the run generation process is executed. (See FIG. 24 through FIG. 27.) If the selection is to "Save As" a template for the run generation process is also executed but the generated files are placed in the template directory. If the selection is to save as a banner or custom application, those absolute screen dimensions are sent to the build engine and its internal database is updated and the run generation process is executed.
- 2: "New" at 144. A test is made by the panel's JavaScript code to see if any user input has been processed and not saved. If so, the user is asked whether this data should be saved. If so, and if the selection is to "Save" as a web page, the build process is completed and the run generation process is executed as described above. If the selection is to "Save" as a template the run generation process is executed but the generated files are placed in the template directory as described above. The panel's JavaScript code then reinitializes its database and calls a build engine method that reinitializes the build engine database.
- 3: "Close" at 145. A test is made by the panel's JavaScript to see if any user input has been processed and not saved. If so, the user is asked whether this data should be saved. If so, and if the selection is to "Save As" a web page, the build process is completed and the run generation process is executed. If the selection is to "Save As" a template the run generation process is executed but the generated files are placed in the template directory. The panel's JavaScript then terminates the build process.
- 4: "Open" at 146. A test is made by the panel's JavaScript to see if any user input has been processed and not saved. If so, the user is asked whether this data should be saved. If so, and if the selection is to "Save As" a web Page, the build process is completed and the run generation process is executed. If the selection is to "Save As" a template the run generation process is executed but the generated files are placed in the template directory. The panel then initiates the dynamic web page resizing technology as described in FIG. 22 below for the open re-initialization mode.
- 5: "Apply" at 147. A template is applied to the existing web site that is being processed by the build engine. The web page and style record definitions of the template replace those of the existing web site. The web page objects of the template are added to the web page objects of the existing web site.
- 6: "Web Site" at 148. The web designer can define settings that will be applied to all web pages in the web site. In one implementation, the web site applications supported include:
  - a: web page. The web page height can be set, as a percentage, larger than the browser window for long vertically scrolled web pages.
  - b: Standard banner sizes.
  - c: Custom. (The user can define any arbitrary web page size and resolution)

US 7,594,168 B2

39

Screen shot FIG. 63 shows the generic web site setting choices presented to the user in one implementation of the invention.

FIG. 21*b* describes the view operations supported by the build engine (29*a* of FIG. 3). In one implementation, the file operations supported include:

- 1: "Normal" at 149*a*. This is the default file mode in which the interface and the build engine are processing user input as described in FIG. 5 through FIG. 23 above.
- 2: "Preview" at 149*b*. The build engine runs the web site off its internal database. The web site will perform in an identical manner as if it had gone through the entire run generation and run time process, but it is being executed on the web designer's computer.
- 3: "Play" at 149*c*. The build engine runs the web site off an external database in a separate browser window. The web site will perform in an identical manner as if it had gone through the entire run generation and run time process, but it is being executed on the web designer's computer.
- 4: "Zoom" at 149*d*. The dynamic web page resizing technology (see FIG. 22 below) is first executed. When the engine is fully reinitialized, and the engine has gone to the current web page, the page and all its objects are drawn to the scale as defined by the zoom level. All object coordinates and sizes are automatically scaled appropriately because they are always defined with virtual screen values, even when the web page is being drawn in the "normal" view.

FIG. 22 describes the dynamic web page resizing technology supported by the build engine. If a user selection of the "Open" command from the "File" menu is detected by the panel at 500, the panel calls an engine method to read selected contents from that web site's external database file.

In one implementation of the invention at 506, the engine reads the web page width and length fields, as well as the background color or background image definition for the first web page of the Web Site. The engine then creates a build engine HTML definition file based on these specifications, and writes this file either to the local disk or the server, depending upon the origination of the build tool.

At 502, if a user completion event occurs inside the web site JavaScript pop-up window, which had been activated when the user selected the "Web Site" command from the "File" menu, the panel determines if the web site page size has been changed. If so, the panel calls an engine method for processing.

Similarly, at 504, if a user selection of a "Zoom" command from the "View" menu is detected by the panel at 504, the panel also calls an engine for processing.

In both the cases at 502 or 504, in one implementation of the invention, the engine writes out a checkpoint record at 508 that is similar to that of a "Websitename".dta "database file" (See FIG. 24). But is given the temporary checkpoint Websitename. The engine then creates a build engine HTML definition file based on these specifications, and writes this file either to the local disk or the server, depending upon the origination of the build tool.

In one implementation of the invention at 510 the engine terminates itself, by stopping all of its threads. Meanwhile the interface writes out four cookies onto the local disk which define the following:

- 1: The re-initialization mode. (Either Open or Checkpoint).
- 2: The current web page number when the resizing event occurred.
- 3: The Web Site Name. (The checkpoint name if in checkpoint mode)

40

4: The zoom level.

The interface then terminates itself by executing the JavaScript "parent.location.href" command, which causes the build engine parent HTML frame file (PFF) to be reloaded (See FIG. 5).

In one implementation of the invention at 512 the re-initialization process begins. The PFF cause both the panel and the build engine to be reloaded and activated. The panel then reads the mode cookie. If the mode is either open or checkpoint, the interface reads the web site name, page number and zoom level cookies, then resets the mode cookie to the initialize state for subsequent operations. The interface then calls an engine method to read the external database, and then to return the necessary values from that database in order to update the interface's database. Finally the engine calls two engine methods in order for the engine to go to the correct current web page and draw that page at the now current zoom level. Normal processing can then resume.

#### 20 Run Generation Process

FIG. 24 through FIG. 27 describe the run generation process. This process starts when the build process detects that the user is finished defining the web site (user has saved the web site and invokes the run generation process), and concludes with the actual uploading of all the necessary web site run time files to the user's server.

FIG. 24 describes the techniques employed by the build engine for the creation of the external database, and the security and optimization techniques that support this process (30 of FIG. 4).

When the panel's JavaScript Code detects a user selection of "Save" or "Save As" from the panel's "File" menu, it draws the appropriate pop-up window and initializes the pop-up window's values for the current web page size as had been defined at FIG. 5 and passed to the build engine. The panel's JavaScript in the "save the web page/template" pop-up window detects a user completion event at 150 (i.e., the designation of a user's web site name followed by the enter key), and calls the appropriate panel JavaScript function. More specifically, after completing the appropriate validity checks, the function calls the appropriate JAVA build engine method, passing as a function call argument the user defined "Websitename". The build engine method checks for the existence of a "Websitename".dta file, and, if so, posts that result into a poll-enabled method return value. The poller checks that value, and if set to true, calls a JavaScript function which draws a pop-up window asking the user to confirm whether the existing web site definition should be overwritten or not. This JavaScript function also calls an appropriate build engine method to reset the return value to false in order to be initialized for the next possible "Save" operation.

Once this verification process is completed the build engine begins the external database creation process at 151, which will vary depending upon the security manager of a given browser at 152. See FIG. 5 for a detailed description of the browser security manager alternatives. If the browser's security manager allows for local disk file creation, the build engine calls a method, which asserts the necessary security policy permissions to create and write a file. If not, the build engine calls the necessary method to create and write a file on the user's server.

The external database contains, as its first record, a "Header" record, which contains can include the following information:

- 1: A file format version number, used for upgrading database in future releases.

## US 7,594,168 B2

41

- 2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user at FIG. 5.
- 3: Whether the application is a web site.
- 4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.
- 5: Web page and styles high watermarks.
- 6: The Websitename.

The header records are written at step 153.

During the build process, as new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, at 154, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the external database as described at 156, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

The settings for all paragraph, text button and image styles are then written as a style record at 155 based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist. The font and color objects are serialized as is discussed in greater detail below (See 159 below).

The body of the external database is then written at 156. All Boolean values are written inside a four-dimensional loop at 157. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects ((i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of line segments per paragraph line and contains the values used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment.).

42

All integer values are written inside a four-dimensional loop at 158. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

Multimedia objects are written inside a two-dimensional loop at 159. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other objects on a web page.

String records are written inside a four-dimensional loop at 160. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The third loop contains values for paragraph line related data, and the innermost loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized.

Single and double floating-point, and long integer records are written inside a two-dimensional loop at 161. The outer loop may be empty. The inner loop contains mathematical values required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

FIG. 25 describes the techniques used to create a customized and optimized run time engine by the build engine (31 of FIG. 4). A versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted at 162. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation at 163.

All external image, video and audio files are resolved at 164. The external references can be copied to designated directories at 164, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries from Sun, Microsoft and Netscape are either installed on the local system (See FIG. 5) or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code at 165. Finally, the run time engine for the web site is created at 166. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file (See FIG. 27).

FIG. 26 shows the techniques used to create the HTML Shell File (HSF) (32 of FIG. 4).

The first step of the process at 167 is to determine whether the dynamic web page and object resizing is desired by testing the application setting, set by the user at FIG. 21a, or possibly

US 7,594,168 B2

43

reset at FIG. 24. If the application was a web page, and thus requiring dynamic web page and object resizing, virtual screen resolution settings, calculated at FIG. 24 at 153, are placed in an appropriate HTML compliant string at 168. If the application is a banner or other customized application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values at 169.

An analysis is made for the background definition for the first internal web page at 170. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the browser. More specifically, if the application required dynamic web page and object resizing (See 167) then JavaScript and HTML compliant strings are generated at 171 so that, when interpreted by the browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated at 172 in order to enable JavaScript to SRS applet communication. In one implementation, the code is generated by performing the following functions:

- 1: Determine the current browser type.
- 2: Load the SRS from either a JAR or CAB File, based on browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.
- 5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated at 173 that perform the following steps at the time the HSF is initialized by the browser:

- 1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.
- 2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the External Database created at FIG. 24.
- 3: Generate an HTML complaint string, dependent upon the type of browser, which causes the current browser to load either the JAR or the CAB File(s).
- 4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

At 174, writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "Websitename".html file is created.

FIG. 27 describes the processes for creating the CAB and JAR Files (33a of FIG. 4). The image objects, if any, which were defined on the first internal web page are analyzed at

44

175. If they are set to draw immediately upon the loading of the first web page, then they are flagged for compression and inclusion in the CAB and JAR Files. The feature flags are analyzed at 176 to determine which JAVA classes have been compiled (See FIG. 25). These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created at 177 that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "Websitename".class, customized run time engine file, and the "Websitename".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "Websitename".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "Websitename".bat and "Websitenamelib".bat files are written at 178. The "Websitename".bat and "Websitenamelib".bat files are then executed under DOS, creating compressed "Websitename".cab and "Websitenamelib".cab files and compressed "Websitename" jar and "Websitenamelib" jar files at 179. The HTML Shell File and the JAR and CAB files are then, either as an automatic process, or manually, uploaded to the user's web site. This completes the run generation processes.

#### Run Time Process

The run time process is shown in FIG. 28 through FIG. 36.

FIG. 28 shows the web page size generation technology utilized by the run time engine. A web surfer points a browser at the HTML shell file (HSF) at 180. The browser begins to interpret the HTML and JavaScript code in the HSF that was created (See FIG. 26). The browser draws either the background color or background image pattern, as defined by the HTML complaint code (See FIG. 26) at 170. The browser then executes the HSF's JavaScript initialization code, which "sniffs" the current browser at 181 to determine its type, and then generates the appropriate HTML code for that particular browser to interpret. This code defines whether the executable files and database will be extracted from inside a compressed CAB file or a compressed JAR file and its location.

Based on the user application (defined at FIG. 21a, or possibly reset at FIG. 24), the HSF at 182 will then execute an appropriate JavaScript function (as created in FIG. 26 at 167). If the application required dynamic resizing of the web page's dimensions, JavaScript code is called which generates HTML code using the JavaScript "document.write" function, which causes the SRS applet to be immediately executed by the browser at 183. The JavaScript code then goes into a timer loop, checking on when the SRS applet is alive before initiating any communication.

After detecting that the SRS has been initialized, a JavaScript function calls the appropriate SRS applet methods at 185, which return the width and height, in pixels, of the current browser window. JavaScript Code is then called which converts the screen resolution independent window width and height values into absolute pixel values. A JavaScript function is then called which use the JavaScript "document.write" function to generate HTML code that define the run time engine specifications, etc. (see FIG. 26) and cause the browser to immediately execute the run time engine. If the application had not required dynamic resizing of the web



US 7,594,168 B2

45

page's dimensions, then a JavaScript function is called which generates HTML code using the JavaScript "document.write" function that defines the fixed dimensions for the web page size and cause the browser to immediately execute the run time engine at 184.

FIG. 29 shows the techniques employed by the run time engine to read the external database and to generate the necessary web page objects (35 of FIG. 4). The run time engine reads a "PARAM" value at 186, from HTML Code that was generated above (see FIG. 26), which points to the "Website-name".dta external database that is compressed into the JAR or CAB File (that was loaded and accessed in FIG. 28). The run time engine then initiates the read operation. In one implementation, the read technique is always non-privileged. If permitted by the current browser as a non-privileged operation, the "Website-name".dta file will be extracted and read from the CAB/JAR file residing in temporary local storage. If not, the run time engine A will read the "Website-name".dta file directly from the server. The header record is read at 187. Any objects, such as fonts and colors, are cast into their original form. The high watermark values, as they are encountered in the header and in the body of the database, are immediately used for setting the limits for the subsequent multi-level read loops for reading in the style record and the web page(s) and object(s) definitions. The virtual screen resolution values are read for the subsequent dynamic resizing of the web page objects.

The style record is read based on its high watermarks, and processed at 188. The definitions for all paragraph, text button, image or other styles are read and stored for subsequent initialization and processing of all paragraph, text button, image or other objects. The data representing the values for the first web page and all its objects is read at 189. The Boolean, integer, string and floating point fields for the first web page are initialized. The serialized multimedia objects for the first web page are read and cast into their final form. (See FIG. 24)

If external files, such as image, audio and video files, must be read as part of the first web page's generation, exception handling routines are executed at 190, as necessary, in the event of any processing errors. In one implementation, error recovery at this stage places the highest priority on a graceful operation cancellation, rather than a web page crash. In the worst case, a particular image, sound or video file may not be available to the web surfer. All other aspects of the web page will likely be available even in this error scenario.

Process step 191 is executed simultaneously with the generation of all the other web pages at 192 by means of multi-programming utilizing thread technology. Thus the first web page will be drawn and active for user viewing and user interaction long before the data for all the other web pages have been read, processed, and initialized. The data representing the values for the subsequent web pages and all their objects are read at 192. The Boolean, integer, string and floating point fields for these web pages are initialized. The serialized multimedia objects for these web pages are read and cast into their final form. (See FIG. 24)

FIG. 30 shows the scaling techniques employed by the run time engine for web page generation (36 of FIG. 4). The first step in the scaling process is to calculate the coordinates that define the origin for the placement of each object for a given web page. (This is usually the upper left corner of the object, defined in actual screen pixels.) A test is made at 193 to determine if the centering attribute is set for the object. If not, the left and top coordinates are converted from the virtual screen values to the local screen values, based on the local screen window resolution at 194. In one implementation,

46

multiplying the virtual coordinate by the local screen window resolution and dividing by the virtual screen resolution determine the conversion.

If the centering attribute is on, then a calculation for the object's width is performed. See processes 197, 198, and 199 below for a description of this calculation. Based on this calculated width, and based on the local screen window resolution, the left coordinate is calculated at 195. One algorithm that can be used is to subtract the screen width, as calculated in 197-199 below, from the local screen window resolution, and divide that result by 2. The top coordinate is calculated the same as in process 194 above.

Based on the object type, determined at 196, a different scaling technology is employed.

If the object is a text button object at 197, the text button object itself, including its background, is not scaled. The virtual width and the local screen width remain the same. However, if a 3D Frame effect is defined, it is scaled based on the following algorithm: if the text string's orientation is Left to Right, the inner width of the 3D Frame, and its placement relative to the text string, is calculated as the length of the text string, plus  $\frac{1}{2}$  of an "n" space on each side, plus an additional offset appended to the right of the inner width to compensate for the italic font style, if defined for the font of that text string. The italic offset can be defined as the font size for the text string, divided by 10, plus 1. The inner height of the 3D Frame can be defined as the font height plus 2 pixels. The font height equals the font's leading plus its ascent plus its descent specifications. The inner height origin can equal the text string origin. The style of the 3D effect (i.e., either a 3D raised look or a 3D depressed look), plus the inner width and height, is sent to a 3D frame build method for the construction of the 3D frame. The width of the 3D frame in pixels can be calculated as the inner width divided by 10 plus 3.

If the text string's orientation is vertical, the inner width of the 3D Frame is an "m" space. The inner height of the 3D Frame can be calculated as the font height times the number of characters in the text string. Both the left and top placement of the 3D frame can be set to the left and top origin of the text string. The width of the 3D Frame can then be calculated as the inner height divided by 10, plus 3.

If an animation is assigned to the text string, the font size used for the initial calculation of the 3D frame is the same as that used to define the animation's initialization value. If the object is a paragraph at 198, and the scaling attribute is on, the maximum width for the paragraph can be defined by the attached paragraph style (or paragraph override) as a percentage of the screen width. This screen width percentage can be converted into an actual width in pixels, based on the local screen's window resolution. If the current screen resolution is the same as that used by the web designer, then the paragraph line end values (just read from the external database) are used without adjustment, bypassing the entire paragraph reformat process. If the current screen resolution is different than that of the virtual screen resolution, then a very compact method of reformat is called (relative to the build engine reformat methods at FIG. 6 and at FIG. 18), and the text for the paragraph is reformatted based on this width.

The run time engine's reformat technology begins by creating one paragraph line for the entire text string assigned to the paragraph text area. All the style overrides are renumbered sequentially with the style records or the non-marked text strings ignored. A simplified "Overflow" reformat method can be called, which chops up the single paragraph line first into paragraph line segments, where each word is defined as a line segment. Because of the word oriented style override architecture, the style overrides have a one-for-one corre-

US 7,594,168 B2

47

spondence with the line segments. Each paragraph line break can be calculated by relying on the simplified word oriented style override technology described above. The paragraph line can be built inside a tight word-by-word loop, with a simple logic check for a style override or hard line break. The paragraph width is then derived as the width of the longest line of the reformatted paragraph, while the paragraph height is defined as the font height times the number of lines. If a 3D frame was defined for the paragraph, it can be scaled based on the following algorithm:

The inner width is defined as the same as that of a text string, but the width of the text string for the longest line is used. The same "n" space and italic offset calculations are used. The inner height is calculated as the font height times the number of lines plus 2 pixels.

If the object is a paragraph, and the scaling attribute is off, then the paragraph is treated the same as a text button object, with the only exceptions that there is no vertical orientation, and the height and width of the 3D frame, if defined, is calculated using the same algorithm as was used for the scaled paragraph above.

If the object is an image at 199, and the scaling attribute is on, the image width can be calculated as the virtual width times the local screen window width divided by the virtual screen width. The image height can be calculated as the virtual height times the local screen window height divided by the virtual screen height. If the image had been resized or rotated, then the virtual width and height of the image would differ from that of that of the original image. If a 3D frame is defined for the image, it can be scaled based on the following algorithm:

The inner width and the inner height of the 3D frame will coincide exactly with the outer edges of the image, after the image had been scaled. Adding the scaled image height to the scaled image width and dividing the result by 40 and adding 3 can calculate the width in pixels of the 3D frame.

If an animation is assigned to the image, then the animation's initialization values for the image's width and height can be used to calculate and draw the initial 3D frame. The coordinates and sizes for the backgrounds for text button, image and paragraph objects can be calculated using the same algorithms as was employed for the calculation and placement of the inner width and inner height for the 3d frame for each object.

FIG. 31 through FIG. 35 shows the multilevel web page and object thread technology employed by the run time engine. The description includes all the animation technologies, transformation technologies, time line technologies and drawing technologies that support this multilevel architecture.

FIG. 31 describes the initial processes for the invention's multilevel web page and object thread technology employed by the run time engine (37 of FIG. 4). Upon the completion of the processing of all the data definitions for the first internal web page (FIG. 30), the main web page thread is created and executed. This causes the run method for the main run time engine class to be executed simultaneously with the reading, processing, and scaling of the data for the subsequent web pages (See FIG. 29). In addition, the reading of any image files defined for the first web page is also performed simultaneously, under the control of an image observer (See FIG. 12). The main run method enters a web page counter loop at 200, the loop being defined from the first internal web page to the high watermark that was set to the number of existing internal web pages for the web site.

A check is made at 201 to see if the current web page exists. If the web page does not exist, and the current web page

48

number is less than that of the high watermark, then the web page counter is incremented by one and the web page counter loop is reentered. If the current web page number equals the high watermark at 202, then the web page counter is reinitialized to the first web page, so that the web page loop may repeat itself, from the first internal web page, depending upon the delay setting for the last web page.

A test is then made on all objects defined for this web page at 203, utilizing a loop whose range is defined by the number of objects per web page high watermarks. More specifically, within this universe of possible objects, if the object exists, and it is defined by a time line in which there is a delayed entrance, then a boolean flag is set for those objects that causes the draw system to suppress drawing these objects during the web page transition as defined below.

A test is then made to determine if the web page has a transition animation defined at 204. If not, the draw system is called for the first time. The draw system for a given web page utilizes a loop whose range is defined by the number of objects per web page high watermarks. The draw system can also employ technology so that the draw process generates a screen image in one or more off-screen buffers, only drawing to the screen when the screen image, or the clipping area for the screen, has been fully generated. This greatly reduces, if not totally eliminates, any screen flicker, and creates visually smooth animation effects.

The first draw function is to draw the web page background into the primary off-screen buffer. The web page background color is drawn, as defined initially at FIG. 21a, or modified for that particular web page at FIG. 20. A test is then made to determine if the web page has a background image pattern, as defined initially at FIG. 21a, or modified for that particular web page at FIG. 20. If it does, and the image observer reports that the image is ready to be drawn, a background image draw loop is executed, defined by the height and width of the background image, and the screen resolution of the current browser window. In the unlikely event that the background image pattern is not yet available, there is a delay until the image observer reports the completion of the image processing operation. The tiled background image pattern is also drawn into the primary off-screen buffer, completely over-drawing the background color. The backgrounds for all non-suppressed (See 203) parent web page text button and paragraph objects are then drawn into the primary off-screen buffer, unless a background transparency flag has been set (See FIG. 7).

The text strings for non-suppressed parent web page text button and paragraph objects are then drawn into the primary off-screen buffer. These text strings are drawn based on their font name, style, size, effect(s), and color. If a paragraph line string, the string may have multiple string segments, each with their own font name, style, etc. If the text button object has its vertical attribute set to true, then the draw system executes a loop defined by the number of characters defined in the text button object. The top and left origin coordinates were set in the usual way (See FIG. 30), but the top coordinate is adjusted by the font height for each iteration of this draw loop. The intelligent 3D Frame, if defined, is then drawn into the primary off-screen buffer for the paragraph and text button objects (See FIG. 30). The primary image objects for the web page are then processed by the draw system. If the image observer reports that the image is ready to be drawn, it is drawn into the primary off-screen buffer, based on the coordinates and size as defined in FIG. 30. If not ready, there is a delay until the image observer reports the completion of the image processing operation. The Intelligent 3D frame, if

US 7,594,168 B2

49

defined, is then drawn into the primary off-screen buffer for the image objects (See FIG. 30).

The draw system is responsive to two other technologies at this stage. The first is user interaction based on the location of the mouse cursor and any user initiated mouse event. This subject will be described in greater detail below in association with FIG. 36. The second is object animation for non-delayed web page objects. This subject will be described in greater detail below in association with FIG. 33.

If the web page transition test at 204 was true, then the run time engine's main run method executes the web page transition animation technology at 205.

FIG. 32 describes the web page transition animation technology. First a lock is placed on this method at 212, as a safety precaution to prevent any interference from other threads during the animation. A test is then made on whether the transition animation setting (See FIG. 20) for the web page is random at 213. If so, a random transition number is generated at 214. The web page thread then begins a particular animation loop at 215, depending upon the random number that was generated at 214 or by the transition animation that was set previously (at FIG. 20). In one implementation, 13 different transition animations plus random are supported including, They are: Fade In, Zoom In, Zoom Out, Zoom to Upper Left, Zoom to Lower Right, Rotate to the Left, Rotate to the Right, Rotate Bottom to Top, Rotate Top to Bottom, Slide to the Left, Slide to the Right, Slide Bottom to Top, and Slide Top to Bottom.

For all web page transition animations, the X and Y animation increment values are calculated by dividing the current browser's screen width and height by the user defined animation resolution at 215. In all animation and draw loops, the number of loops can equal the number of animation frames as set at FIG. 20. The timer delay for all animations, in milliseconds, can be calculated by dividing the number of frames per second (See FIG. 20) into 1,000.

For a description of "Fade In" Technology see FIG. 33. A "Zoom In" algorithm sets the initial scaled width and height for the current web page image to zero and the prior web page image to its full size. In each animation and draw loop the previous web page's final image state is drawn into a secondary off-screen buffer at 216. (If this is the first occurrence of the first web page, then the secondary off-screen buffer is set to the background of the first web page.) The upper left hand corner (origin) of the current web page can be calculated based on the following formula: browser screen width minus the scaled width divided by two.

The scaled image of the current web page is then drawn into the secondary off-screen buffer at the calculated origin, using the current scaled width and height for the web page image. This merged image of the prior and scaled version of the current web page is then drawn to the screen.

A timer delay then occurs as defined at 215, after which the X and Y animation increment values are added to the scaled width and height for the current web page image. The animation loop is then repeated to its conclusion at 218.

The other eleven web page transition animations follow a similar methodology, but have quite different calculations, which are based on the following variables:

- 1: Order of drawing of the prior and current web pages.
- 2: Initialization values for the X and Y origin coordinates for the current and prior web pages.
- 3: The initial values for the scaled width and height for the current and prior web pages.
- 4: Whether X and Y origin coordinates for the current and prior web pages increment, decrement, or remain the same.

50

- 5: Whether the values for the scaled width and height for the current and prior web pages increment, decrement, or remain the same.

For the "Zoom Out" animation, the current page is drawn first and always drawn at 100%. The prior web page is initialized also at 100%, but its X and Y origin coordinates are incremented and its scaled width and height values are decremented, by the appropriate values, for each animation iteration.

For the "Zoom to Upper Left", "Zoom to Lower Right", "Rotate to the Left", "Rotate to the Right", "Rotate Bottom to Top" and "Rotate Top to Bottom" animations, a common data initialization and data increment strategy is implemented.

- 1: The X and Y variables for page image one is set to zero.
- 2: The X and Y variables for page image two is set to the right and bottom edges of the browser window.
- 3: The scaled width and height variables for page image one is set to 100% of the browser window's resolution.
- 4: The scaled width and height variables for page image two is set to zero.
- 5: For each loop iteration, the scaled width and height variables for page image one are decremented by the X and Y animation increment values defined at 215.
- 6: For each loop iteration, the scaled width and height variables for page image two are incremented by the X and Y Animation increment values defined at 215.

For the "Zoom to Upper Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. (upper left corner of the browser window) Its scaled width and height values are always set to the current values for scaled width and height variables for page image one. The X and Y origin coordinates for the current web page can be calculated by subtracting the current values of image two's scaled width and height variables from the initial values of the X and Y variables for page image two. The scaled width and height values for the current web page can be set to the current values for the scaled width and height variables for page image two.

For the "Zoom to Lower Right" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width and height values are always set to the current values for scaled width and height variables for page image two. The X and Y origin coordinates for the prior web page are set to current values of image two's scaled width and height variables. The scaled width and height values for the prior web page are set to the current values for the scaled width and height variables for page image one.

For the "Rotate to the Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to current value of image one's scaled width variable. Its scaled height value is always set to the bottom of the browser's window. The X origin coordinate for the current web page can be calculated by subtracting the current value for image two's scaled width variable from the initial value for image two's X origin coordinate. The Y origin coordinate for the current web page is always set to zero. Its scaled width value is set to current value of image two's scaled width variable. Its scaled height value is always set to the bottom of the browser's window.

For the "Rotate Bottom to Top" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to the width of the browser window. Its scaled height value is set to current value of image one's scaled height variable.

The current web page's X origin coordinate is always set to zero. The Y origin coordinate is calculated by subtracting the current value of image two's scaled height variable from the

US 7,594,168 B2

51

initial value for image two's Y origin coordinate. Its scaled width value is always set to the right edge of the browser's window. Its scaled height value is set to current value of image two's scaled height variable.

For the "Rotate Top to Bottom" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to the width of the browser window. Its scaled height value is set to current value of image two's scaled height variable.

The prior web page's X origin coordinate is always set to zero. The Y origin coordinate is set to the current value of image two's scaled height. Its scaled width value is always set to the right edge of the browser's window. Its scaled height value is set to current value of image one's scaled height variable.

For the "Slide to the Left", "Slide to the Right", "Slide Bottom to Top" and "Slide Top to Bottom" transition animations, a common data initialization and data increment strategy is implemented. The strategy includes:

- 1: The X and Y variables for page image one is set to zero.
- 2: The X and Y variables for page image two is set to the right and bottom edges of the browser window.
- 3: For each loop iteration, the X and Y variables for page image one are incremented by the X and Y animation increment values defined at 215.
- 4: For each loop iteration, the X and Y variables for page image two are decremented by the X and Y animation increment values defined at 215.
- 5: The scaled width and height values always remain at 100% of the browser windows width and height.

For the "Slide to the Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. The current web page's X origin coordinate is set to the current value of page image two's X variable. Its Y origin coordinate is always set to zero.

For the "Slide to the Right" Animation, the current web page is drawn first, with its X and Y origin coordinates always set to zero. The prior web page's X origin coordinate is set to the current value of page image one's X variable. Its Y origin coordinate is always set to zero.

For the "Slide Bottom to Top" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. The current web page's Y origin coordinate is set to the current value of page image two's Y variable. Its X origin coordinate is always set to zero.

For the "Slide Top to Bottom" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. The prior web page's Y origin coordinate is set to the current value of page image one's Y variable. Its X origin coordinate is always set to zero.

After the last animation cycle is completed for any of the transition animations at 218, the animation process is unlocked, and process step 206 shown in FIG. 31 is then executed.

Returning to FIG. 31, the main web page thread's run method then executes a text button and image object time line, transformation and animation loop at 206. This range loop is defined from the first object on the given web page to the high watermark for the number of those objects on a web page for this web site. A test is made on each object on whether an animation, transformation and/or time line has been assigned at 208.

If so, an "instance" of the time line class for that particular object type is created at 209. An "instance" of a class is a fundamental aspect of object oriented programming (OOP). Each time, the line class is implemented with a "runnable" interface, so that they can be executed as independent threads.

52

Communication of data, between the "instance" of a class and the main run engine class can be accomplished in OOP using several different techniques. In one implementation, this construction, passed as an argument, is used to permit different objects to address each other's variables and databases. A thread is then created, utilizing a two-dimensional object internal database architecture (web page number by internal object number). This methodology is convenient for permitting all object time lines for a given web page to be managed and synchronized. The object's thread is then "started".

The result of this process at 209 is that an independent thread has been created for each appropriate object on a given web page, all executing simultaneously with each other and with the main run time engine web page thread, subject to the definitions of their independent time lines at 210. See FIG. 33 for a description of the time line technology. When the main web page thread has finished the text and image loop at 207, the draw system is activated; the run time engine can now respond to user interactions, and the main web page thread transitions into a "Join" loop at 211. See FIG. 35 for a description of this process.

FIG. 33 shows the time line technology used by the run time engine. The techniques and algorithms employed to create this technology permit each web page object to have an independent yet synchronized existence with each other, with the main web page thread, and with child objects that each main or parent object may spawn. Furthermore, each object and each of their child objects are capable of performing multiple animations and transformations, either serially or simultaneously. Database initialization is first accomplished for each object thread. This assures that the object thread's database is set to the correct initial values as required for that particular object, and that the references to the main web page thread's database are established.

A test is then made to determine if the object has a time line definition assigned to it at 219.

If not, a test is made at 220 on certain two-dimensional object definition variables in order to determine which of the following four states have been defined for the object: animation without a transformation; transformation without animation; animation, with the transformation occurring simultaneously with the animation; and animation and transformations occurring in a serial manner.

If the test shows that the object has an animation defined, but no transformation, then certain two-dimensional status variables are set, and an "instance" of the "animation class" for that particular object type is created at 229. Each "animation class" is also implemented with a "runnable" interface. An object animation thread is then created, utilizing the two-dimensional object internal database architecture (See FIG. 8). This object animation thread is then "started". Communication between the object animation thread, the parent time line thread, and its parent, and the main web page thread, are accomplished as discussed in process 209. The object time line thread then executes a "Join" method. This puts the object time line thread in a "wait state". When the thread it is waiting for is completed, this child thread "joins" the parent object time line thread, and the object time line thread then continues its process. Other forms of synchronization between two independent threads could have been implemented as is known in the art.

The techniques employed at 229 to implement object animation vary by object type. In one implementation, for text button object animations, 26 different animations are supported including: Zoom In, Zoom Out, Grow NW, Grow NE, Grow SE, Grow SW, Shrink SE, Shrink SW, Shrink NW, Shrink NE, Enter N, Enter NE, Enter E, Enter SE, Enter S,

US 7,594,168 B2

53

Enter SW, Enter W, Enter NW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW. In one implementation, for image object animations, 29 different animations are supported including: Fade In, Fade out, Rotate, Zoom In, Zoom Out, Grow NW, Grow NE, Grow SE, Grow SW, Shrink SE, Shrink SW, Shrink NW, Shrink NE, Enter N, Enter NE, Enter E, Enter SE, Enter S, Enter SW, Enter W, Enter NW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW.

As discussed above with regard to FIG. 17, each animation type has a defined speed, resolution, and number of animation cycles. These settings are stored in the main web page class, and are passed to the particular animation thread through a two-dimensional object internal database architecture as discussed in process step 209 above during the animation thread's initialization process. The animation thread then executes, in its run method, a main animation loop that has the number of iterations set to the end number of animation cycles, as assigned to that particular text button object.

Text button animations are currently implemented in three logical groups. Group One includes "Zoom In", "Grow NW", "Grow NE", "Grow SE", and "Grow SW". Group Two includes "Zoom Out", "Shrink SE", "Shrink SW", "Shrink NW", and "Shrink NE". Group Three includes "Enter N", "Enter NE", "Enter E", "Enter SE", "Enter S", "Enter SW", "Enter W", "Enter NW", "Exit N", "Exit NE", "Exit E", "Exit SE", "Exit S", "Exit SW", "Exit W" and "Exit NW".

For Group One text button animations, the animation font size is initialized at a very small value, and in one implementation is set at 4 Points. The animation point size increment can be derived by dividing the resolution (number of animation frames) into the font size for that text button object. The run method then executes a secondary animation loop, which will terminate when the animation font size equals the text button object point size. For each secondary animation loop, the length of the current animated text string is calculated, a new font object is created for the current animation point size, and the font metrics for that new font are created. If the text button object has a vertical orientation, the animated text button object's width is calculated to be the width of an "m" space, in the current animated font. The animated text button object's height is calculated to be current animated font height times the number of characters in the text string. If the text had a horizontal orientation, the animated text button object's width is calculated to be the width of the text string in the current animated font. The animated text button object's height can be calculated to be the font height of the current animated font. The calculations for X and Y coordinates for the animated text button object depend upon which animation was defined within the Group One-text button animations. The X and Y animation increments can be calculated utilizing the height and width, in pixels, of the text button object scaled to the current browser's window, utilizing the text button animation resolution, and considering whether the animating text button object is being centered during the animation ("Zoom Out") or not. These calculations are similar to those for the web page transition animations discussed with regard to FIG. 32.

The draw system is then called. Based on the values of the two-dimensional status variables that had been set initially, the draw system executes the appropriate animation draw routine utilizing, through the data communication techniques already discussed, the current animation font point size, and the current animation X and Y coordinates. If a text background is to be drawn, the same algorithm as defined in FIG. 31 is used. If a 3D Frame is assigned, the current animated string width and height are passed to the appropriate 3D frame

54

generation method, and the frame is drawn with the same algorithm as defined in FIG. 31, but utilizing the current animation X and Y coordinates. The text button object's orientation is also handled by the draw system with the same algorithms as defined in FIG. 31.

The text button animation thread then executes a timer delay, whose value had been defined in FIG. 17. When the timer reactivates the text button animation thread after the appropriate delay, an animation cycle completion test is made to see if the text button object's point size minus the animation point size is less than the animation point size increment. This type of testing methodology permits the invention to utilize integer values, as opposed to floating point values, for the text button animation. This improves the execution of the animation considerably.

If the above test is true, the animation point size is set equal to the object point size and a final call is made to the draw system for that animation cycle. A test is then made to see if the current animation cycle equals the total number of animation cycles as defined in FIG. 17. If not, a new animation cycle is initiated, with the animation values reinitialized. If this was the last animation cycle the text button animation thread calls its "stop" method, which sets the required status variables as appropriate, then terminates itself. This causes the parent text button time line thread to be reactivated through the "join" mechanism.

If the results of animation cycle completion test are false, the current animation point size is increased by the animation point size increment. A new font object is created for the now current animation point size, and new font metrics for that new font are created. If the text button object has a vertical orientation, the animated text button object's width is calculated to be the width of an "m" space, in the now current animated font. The animated text button object's height is calculated to be the now current animated font height times the number of characters in the text string. If the text has a horizontal orientation, the animated text button object's width is calculated to be the width of the text string in the now current animated font. The animated text button object's height is calculated to be the font height of the now current animated font. The calculations for the new X and Y coordinates for the animated text button object are then completed, as appropriate, and the draw system is called again.

The algorithms for Group Two text button animations are very similar to those of Group One. The differences are just in what are the initial animation values, and whether the animation point size increments and the animation X and Y coordinate increments are added or subtracted from the then current animation point size and the then current X and Y coordinates for the animating text button object.

For Group Three text button animations, the distance that the text button animation will move is calculated, in pixels, from its initial location to its final location in the current browser window. The X and Y animation increments are calculated by dividing that distance by the resolution of the text button animation. All the other algorithms for Group Three text button animations are generally a subset of those for Group One, and similar to the web page slide transition animations defined with reference to FIG. 31.

Referring again to FIG. 33, image animations at process step 229 can currently be grouped into five logical classes. As with text button animations, Group One includes "Zoom In", "Grow NW", "Grow NE", "Grow SE", and "Grow SW". Group Two includes "Zoom Out", "Shrink SE", "Shrink SW", "Shrink NW", and "Shrink NE". Group Three includes "Enter N", "Enter NE", "Enter E", "Enter SE", "Enter S", "Enter SW", "Enter W", "Enter NW", "Exit N", "Exit NE",

US 7,594,168 B2

55

“Exit E”, “Exit SE”, “Exit S”, “Exit SW”, “Exit W” and “Exit NW”. In addition, image animations have a Group Four, which includes “Fade In” and “Fade Out”. Group 5 image animations include the “Rotate” Animation.

For Group One image animations, the animation width and height increments are calculated by dividing the image object’s width and height by the resolution (number of animation frames) as set in FIG. 17. The initial animation width and animation height values are set to a very small number, currently equal to the animation width and height increment values just calculated. The calculations for X and Y coordinates for the animated image object depends upon which animation was defined within the Group One text button animations. The X and Y animation increments are calculated utilizing the height and width, in pixels, of the image object scaled to the current browser’s window, utilizing the image animation resolution, and considering whether the animating image object is being centered during the animation (“Zoom Out”) or not. These calculations are similar to those for the web page Transition Animations discussed above with regard to FIG. 32.

The run method then executes a secondary animation loop, which will terminate when the animation width equals the image object’s width. The algorithms employed by the invention to change the animating object’s height, width, X coordinate, and Y coordinate are very similar to those employed for Group One text button animations, and will not be repeated here. The techniques to utilize the draw system for drawing the image animation, the time delay technique, and the post draw logic tests and actions are also very similar.

The algorithms for Group Two image animations are very similar to those of Group One. The differences are just in what are the initial animation values, and whether the animation width and height increments and the animation X and Y coordinate increments are added or subtracted from the then current animation width and height and the then current X and Y coordinates for the animating image object.

For Group Three image animations, the algorithms are identical to those of Group Three text button animations. For Group Four image animations, the “alpha” value of a given image object is utilized in order to implement “Fade In” and “fade Out” animations. The alpha value can range from 0 to 255, depending upon the image strength desired. The value for an alpha animation increment variable can be calculated by dividing the resolution of the animation into 255, after making the necessary adjustments to keep the data in integer form, without losing resolution due to integer rounding errors. For a “Fade In” animation the value of an alpha animation variable is set to zero. The run method then executes a secondary animation loop, which will not terminate until 255 minus the then current value of the alpha animation variable is less than the value alpha animation increment variable. A “Fade In” image filter can be created for each iteration of the animation loop, using the current setting of the alpha animation variable. An image producer can also be created with pointers to the last image bitmap produced for the image object in the last animation loop and to the image filter that has just been created. The image producer, under the control of a media tracker then creates a new image bitmap. The animation thread then “waits” for the completion of this image-processing event using the media tracker. Upon completion, the draw system is called which draws the then current state of the image object. The image animation thread goes into a timer delay of some preset value (in one implementation 500 milliseconds), to permit a smooth visual animation effect. The value for the alpha animation increment is added to alpha animation variable and the loop is then

56

repeated until the loop condition is met. Then the “stop” method is called, certain status variables are set, and the image animation thread terminates itself. This causes the parent image time line thread to be reactivated through the “Join” mechanism.

The “Fade Out” animation employs very similar technology, except that:

- 1: the alpha animation variable is set to zero,
- 2: the value for the alpha animation increment is subtracted, and
- 3: the loop termination test is when the value for the alpha animation variable is less than the value for the alpha animation increment.

For the Group Five image rotate animation, a different bitmap for the image object is created for each animation frame through the use of a progression of standard geometrical transformations on the original image bitmap. A secondary animation loop is then executed as defined by the number of animation frames. In each loop iteration, an image object is created from an appropriate image bitmap selected from among the set just created, the necessary two-dimensional variables are set to communicate with the draw system, and the draw system is then called. The image animation thread then executes a timer delay method based on the delay setting as defined above with reference to FIG. 17. When the timer reactivates the image animation thread after the appropriate delay, the next iteration of the secondary animation loop is repeated until the loop condition is met. Then the “stop” method is called, certain status variables are set, and the image animation thread terminates itself. This causes the parent image time line thread to be reactivated through the “join” mechanism.

Returning to process step 220 shown in FIG. 33, if the object had a transformation, but not an animation, then certain two-dimensional status variables are set, and an “instance” of the “transformation class” for that particular object type is created at 228. Each “transformation class” is also implemented with a “runnable” interface. An object transformation thread is then created, utilizing the invention’s two-dimensional object internal database architecture. This object transformation thread is then “started”. The inter-thread communication technology and the “join” technology employed for object transformations is the same as for object animations.

If the transformation is being applied to a text button object at 228, then a timer delay method is executed based on the delay setting as described in association with FIG. 18. When the timer reactivates the text button transformation thread after the appropriate delay, the appropriate two-dimensional status variables are set to inform the draw system which state of the current text button object to draw. The draw system is called and, based on the settings for the above mentioned two-dimensional status variables, either the “normal”, mouse over”, mouse down” or “pop-up” states of the text button object’s background, if any, the text button object’s string, and the 3D frame, if any, are drawn. If additional transformations are defined (FIG. 18), the above process is repeated, based on the timer delay and object states defined for the subsequent transformations. When the last transformation is completed, the “stop” method is called, which sets the required status variables as appropriate. This causes the parent text button time line thread to be reactivated through the “join” mechanism.

If the transformation is being applied to an image object at 228, then a timer delay method is executed based on the delay setting (as defined in FIG. 18). When the timer reactivates the image transformation thread after the appropriate delay, image transformation technology is executed. In one imple-

US 7,594,168 B2

57

mentation, the image transformation technology utilizes the “alpha” value of a given image object state in order to fade in and fade out images. The alpha value can range from 0 to 255, depending upon the image strength desired. The value for an alpha transformation increment variable is calculated by dividing the resolution of the transformation into 255, after making the necessary adjustments to keep the data in integer form, without losing resolution due to integer rounding errors. The value of an alpha transformation variable is set to zero. Depending upon the settings as defined in FIG. 18, the bitmap for one image object state is initialized to an alpha value of zero, while another is initialized to an alpha value of 255. The appropriate two-dimensional status variables are set for communication with the draw system.

A transformation loop is then executed, until 255 minus the then current value of the alpha transformation variable is less than the value alpha transformation increment variable. This methodology again keeps all calculations in the form of integers, as opposed to floating point, thus speeding up the transformation process.

Two “Fade In” image filters are created for each iteration of the transformation loop. The first uses an alpha value calculated at the current setting of the alpha transformation variable. The second uses an alpha value calculated at 255 minus the current setting of the alpha transformation variable. Two image producers are also created with pointers to the last image bitmap produced for each image object state in the last transformation loop and to the two image filters that had just been created. The two image producers under the control of two media trackers then create two new image bitmaps. The transformation thread then “waits” for the completion of these two image processing events using the media trackers. Upon completion, the draw system is called which draws the then current state of the two image object states, in the correct order, and in the correct location. The image transformation thread goes into a timer delay of some preset value (500 milliseconds in one implementation), to permit a smooth visual transformation effect. The loop is then repeated until the loop condition is met. Then the “stop” method is called, certain status variables are set, and the image transformation thread terminates itself. This causes the parent image time line thread to be reactivated through the “join” mechanism.

Returning to process step 220 in FIG. 33, if the object was defined with an animation and transformation that would execute in a serial manner, then certain two-dimensional status variables are set, and an “instance” of the “transformation” class for that particular object type is created at 230. An object transformation thread is then created, utilizing the two-dimensional object internal database architecture. This object transformation thread is then “started” and the parent object time line thread “waits” to be “joined”.

If a text button object, then a primary loop is executed, with the number of iterations set to the number of transformations. After the execution and return from a timer delay event, if any, an “instance” of the text button animation class is created, and then a text button animation thread is created and “started”. The parent text button transformation thread then waits to be “joined”. This causes the text button animation thread to be executed, in the manner described at 229. When the text button animation thread completes its execution, it calls its “stop” method, which sets the necessary status variables and then terminates itself. This causes the text button animation thread to “join” the parent text button animation thread, causing that thread to resume processing. The first text button transformation is then executed, in the manner described at 228. After the execution and return from another timer delay event, if any, another “instance” of the text button animation

58

class is created, and then another text button animation thread is created and “started”. The parent text button transformation thread again waits to be “joined”. This causes the text button animation thread to be executed again with the animation being executed, based on the definition set at FIG. 18, on a different text button object state. The loop is then repeated until the last text button transformation is completed. Then the text button transformation thread calls its “stop” method, certain status variables are set, and the text button transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the “join” mechanism.

If an image object, the mechanism of the image transformation thread spawns image animation threads, before each transformation, and is the same as that of a text button object. The actual image transformation process is identical to that described at 228. When completed the “stop” method is called, certain status variables are set, and the image transformation thread terminates itself. This causes the parent image time line thread to be reactivated through the “join” mechanism.

Returning to process step 220 in FIG. 33, if the object was defined with a simultaneous animation and transformation, then certain two-dimensional status variables are set, and an “instance” of the “super transformation class” for that particular object type is created at 231. In one implementation, the animation, transformation, and super transformation classes are integrated into one structure in order to reduce code size and increase execution speed. Each “super transformation class” is also implemented with a “runnable” interface. An object super transformation thread is then created, utilizing the two-dimensional object internal database architecture. This object super transformation thread is then “started”. The inter-thread communication technology and the “join” technology employed for object super transformations is the same as for object transformations.

If a text button object, a calculation is made in order to prorate the text button animation process across the defined text button transformation process. The calculation is driven by the number of text button animation frames, and prorates from that total the number of frames that should be assigned to each transformation state. This can be done by dividing the sum of all the transformation times by each individual transformation time, and multiplying that result by the number of frames, making necessary adjustments to prevent integer rounding error. After these calculations are completed, the text button animation is executed in a similar manner as was defined at 229. However, when the appropriate number of animation frames had been drawn, certain two-dimensional status variables are set prior to calling the draw system for the next animation frame, so that the correct text button object state is drawn, in the correct size and with the correct coordinates, by the draw system. When the super transformation process is completed the “stop” method is called, certain status variables are set, and the text button super transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the “join” mechanism.

If an image object, a calculation is made in order to prorate the image animation process across the defined image transformation process. The calculation is driven by the number of image transformation events that would occur (where each one can be set at approximately 500 milliseconds) over the entire animation event. A calculation is performed in order to calculate how many image transformation events should be assigned to each transformation state. This is done by dividing the sum of all the transformation times by each individual

US 7,594,168 B2

59

transformation time, and multiplying that result by the total number of transformation events, making necessary adjustments to prevent integer rounding error. A calculation is then made to allocate the number of animation frames to each image transformation event. After these calculations are completed, the image animation is executed in a similar manner as was defined at 229. However, when the appropriate number of animation frames had been drawn, the image transformation technology is called to perform the next transformation event. The alpha transformation increment can be defined by dividing 255 by the number of transformation events assigned to that transformation. The draw system is then called. When the number of image transformation events assigned to a given image transformation is reached, then certain two-dimensional status variables are set prior to calling the draw system for the next animation frame, so that the correct image states, in the correct size and with the correct coordinates, are utilized by the draw system. This entire animation/transformation process will be repeated by the number of image animation cycles. When the super transformation process is completed the “stop” method is called, certain status variables are set, and the text button super transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the “join” mechanism.

Returning to process step 219 in FIG. 33, if the object had a time line, then a test is made at 221 on whether an appearance delay had been defined in FIG. 19. If so, a timer event is set at 222.

When the timer reactivates the object time line thread after the appropriate delay, a test is made on whether an entry animation/transformation has been defined for this object time line at 224, as described FIG. 19. If so, based which animation/transformation process was defined, it is created and executed at 228, 229, 230, or 231. In one implementation, 13 entry animations are supported for both text button and image objects, and an additional “Fade In” entry animation is supported for image objects. The 13 common entry animations supported include Zoom In, Grow NW, Grow NE, Grow SE, Grow SW, Enter N, Enter NE, Enter E, Enter SE, Enter S, Enter SW, Enter W and Enter NW.

If no entry animation/transformation is defined, or when the entry animation/transformation has “joined” the object time line thread, a test is made to determine if any child time lines have been defined at 225, as described in FIG. 19, for this parent object time line. If so, an “instance” of the “child time line class” for that particular object type is created at 226. Each “child time line class” is also implemented with a “runnable” interface. An object child time line thread is then created, utilizing the two-dimensional object internal database architecture. This object child time line thread is then “started”. The inter-thread communication technology and the “join” technology employed for object child time lines is the same as for object time lines. Either a text button child time line thread or an image child time line thread, or both, can be spawned at this time.

Simultaneous with the execution of any spawned text button child time line threads, the parent object thread then executes the defined main animation and/or transformation. As with non-time line object threads, a test is made on certain two-dimensional object definition variables in order to determine which of the following four states have been defined for the object at 227: animation without a transformation; transformation without animation; animation, with the transformation occurring simultaneously with the animation; and, animation and transformations occurring in a serial manner.

Based on the results of this test, an appropriate “instance” of an appropriate animation, transformation, or super trans-

60

formation class is created, and an appropriate animation, transformation, or super transformation thread is created and “started”. This results in the execution of process steps 228, 229, 230, or 231, as defined above.

The parent object time line thread then executes a “join” method. This again puts the object time line thread in a “wait state”. When the thread it is waiting for is completed, the child thread “joins” the parent object time line thread, and the object time line thread then continues its process.

The object time line thread then checks to see if there is a departure delay defined at 232. If so, it sets a timer event at 233. When the timer reactivates the object time line thread after the appropriate delay, a test is made at 234 on whether an exit animation/transformation has been defined for this object time line, as described in FIG. 19. If so, it is created at 235, and performed as discussed with reference to processes 228, 229, 230, or 231. In one implementation, 13 exit animations supported for both text button and image objects, and an additional “Fade Out” exit animation is supported for image objects. The 13 common exit animations include: Zoom Out, Shrink NW, Shrink NE, Shrink SE, Shrink SW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW.

If no exit animation/transformation is defined, or when the exit animation/transformation has “joined” the object time line thread, the parent object time line thread then executes a “join” method if it had spawned any child time lines. This again puts the object time line thread in a “wait state”. Finally, when then the child time line threads, if any, “join” the parent object time line, the “stop” method for the parent time line is called. Certain status variables are set, and the parent object time line thread terminates itself. This causes the main web page time line that had been in a “join” loop at 211 of FIG. 31, since the invocation of the object time lines, to be “joined” by this particular object time line thread.

FIG. 34 shows the technology employed by the run time engine for implementing child time lines for text button and image objects. Child text button object time lines and child image object time lines are subsets of their parent object time lines. First a test is made at 237 on whether an appearance delay had been defined (See FIG. 19). If so, a timer event is set at 238. When the timer reactivates the child object time line thread after the appropriate delay, a test is made on whether an entry animation has been defined for this child object time line at 239 (as described FIG. 19). If so, it is created and executed at 240 in a manner identical to that described at process step 229 in FIG. 33. The same 13 entry animations supported for parent object time lines are also supported for both child text button and image objects, and the additional “Fade In” entry animation is supported for child image objects. The “join” mechanism described in FIG. 33 is employed in an identical manner at 240 to synchronize the child time line thread with its entry animation thread.

After being “joined” and reactivated, the child object time line performs a test at 241 on whether an exit delay had been defined (See FIG. 19). If so, a timer event is set at 242. When the timer reactivates the child object time line thread after the appropriate delay, a test is made on whether an exit animation has been defined for this child object time line at 243, as described in association with FIG. 19. If so, it is created and executed at 244 in a manner identical to that described above at process step 229 in FIG. 33. The same 13 exit animations supported for parent object time lines are also supported for both child text button and image objects, and the additional “Fade Out” exit animation is supported for child image objects. The “join” mechanism described above in association with FIG. 33 is employed in an identical manner at 245 to synchronize the child time line thread with its parent object



US 7,594,168 B2

61

time line thread. As discussed at process step 236 in FIG. 35, the parent object time lines “wait” until all their child time lines have terminated, before they in turn terminate and “join” the main web page time line at FIG. 35.

FIG. 35 describes technology employed by the run time engine for the web page and object thread loop. As noted in FIG. 31 at process step 211, after all the text button and image time line threads for the current web page had been launched, the main web page thread executed a “join” loop, waiting for the completion of all the parent object time line threads. Because each parent object time line thread waited for their child object time line threads to be “joined”, as well as any other spawned animation threads, transformation threads, and/or super transformation threads, the effect of this “join” loop at 246 is that the web page thread will not resume processing until all parent time line threads have completed and that of all of their spawned threads.

Upon resuming its processing after the “join” process at 246 has been completed, the main web page thread checks at 247 to see whether the current web page has an automatic termination, based on a timer delay, or whether the web page will wait for a user interaction before terminating.

If the web page has a time delay based termination setting, then a timer method is called at 249, and the web page goes to “sleep” awaiting the completion of the timer event.

When the timer event occurs, the web page thread resumes processing by incrementing the web page counter by one, and the entire web page process, which began at process step 200 in FIG. 31, is repeated. If the current web page termination setting was to set to wait until user interaction, then Web page thread is placed in a “pause” state, and the run time engine waits to respond to any mouse, keyboard or other user initiated event.

FIG. 36 describes the technology employed by the run time engine for responding to user interactions. As mentioned in association with process step 204 of FIG. 31, as soon as the draw system has been activated, the run time engine will respond to any user interactions that have been defined (See FIG. 16). This is also true during any object time line events, as with respect to process step 207 of FIG. 31. The run time engine currently responds to “mouse over” and “mouse down” events for text button, image, and paragraph objects. For form objects, the run time engine will also respond to keyboard events. As the full-featured programming languages supported by browsers evolve, the run time engine may be configured to respond to other user interactions, including but not limited to single and double clicks from both the left and right mouse button, voice commands, eye focusing technologies, touch screen technologies, and push technologies originating from a server.

The run time engine invokes a “dynamic mouse to object recognition” technology at 251 in order to be responsive to the following elements:

- 1: The location of objects will vary based on the viewer’s screen resolution and browser window size as discussed above with regard to FIG. 27.
- 2: Objects may move or resize themselves based on time lines and animations.
- 3: Objects may have different sizes based on the state they are being displayed in based on time lines and transformations.
- 4: More than one object can occupy the same screen location, and which objects occupy that location may change over time based on time lines, animations, and transformations.

The run time engine maintains, in its internal database, the object’s current X and Y origin coordinates, and the object’s

62

current width and height, in pixels, based on the current viewer’s screen and browser window size. This can be accomplished by first converting all coordinates and sizes to the current viewer environment with the scaling technology as discussed above with regard to FIG. 27. Every time line, animation, and transformation thread updates, in real time, the run time engine’s internal database positional and size variables of the objects they define, utilizing the data communication techniques described above with reference to FIG. 33.

The run time engine employs mouseEnter, mouseMove, mouseDown, mouseDrag, mouseUp, and mouseExit methods to constantly monitor the state of the mouse at all times. The onClick method (to detect a single click) and a specialized method to detect a double click event are also employed. The onKeyDown method, with processing the returned scan code, permits the run time engine to process all keyboard events. The mouseEnter, mouseMove, mouseDown, mouseDrag, mouseUp, and mouseExit methods return to the run time engine the exact X and Y coordinates of the mouse cursor at the instant that particular mouse event occurred. Thus for each supported mouse based user interaction technique supported by the run time engine, a two-dimensional loop exists (web page number by object number) in which the current bounding rectangle for every object on a given web page is being compared to the current mouse cursor location at all times. The bounding rectangle is simply the current X and Y origin coordinates of an object, extended by its current width and height. In this way, the run time engine is informed if the current mouse cursor location falls within one or more bounding rectangles. Parenthetically, the run time engine also knows when the current mouse cursor falls outside the bounding rectangle of a given object.

Based on the type of mouse user interaction at 252, the run time engine employs different techniques and executes different methods. If the viewer moves the mouse at 253, the mouseMove method informs the run time engine immediately of this event and the current mouse cursor coordinates.

If this user mouse move action caused the mouse to move into one or more bounding rectangles of any text button or image object(s) at 254, or out of one or more bounding rectangles of any text button or image object(s) at 255, then appropriate two-dimensional status variables are set and the draw system is called. The draw system interprets the relevant two-dimensional variables for the existing text button and image object(s) on the current web page in its draw loop as described above with reference to FIG. 16, and draws the correct backgrounds, text strings, images, and 3D frames based on the state of each object, as just set by the “mouse-Move” computational two-dimensional loop. If the mouse has entered into or out of the bounding rectangles of any image and/or text button object that has a defined text button, image and/or video pop-up object (See FIG. 16), then the draw system paints or effectively erases the appropriate background, text string, images and/or 3D frame for these pop-up objects. If the location of any of these text button and image objects or child pop-up objects is changing over time because of time line, animation, or transformation threads, the draw system, as described in association with FIG. 33 and FIG. 34, is aware of these dynamics, and redraws the screen as these real time events occur. If any sound or video events were defined (See FIG. 16) for any text button or image objects, then the run time engine plays those sound and/or video files or channels as defined. As multiple objects can be defined that each have associated sound (and even video) files, and these objects can be overlaid on each other, either completely or partially, very interesting synchronized multiple sound tracks

US 7,594,168 B2

63

can be constructed, in which certain designated sounds are played, or stopped, based purely on user mouse movement.

If the viewer moves the mouse into or out of the bounding rectangle for a paragraph hot link at 256, then appropriate four-dimensional status variables (web page by paragraph number, by paragraph line number by paragraph line segment number) are set or reset and the draw system is called. The draw system paints the background color behind the hot link text string, and the color for the hot link text string in the hot link "mouse over" or the hot link normal colors. The text string may be underlined or in a bold font, depending on the settings.

If the viewer presses a mouse button at 257, the mouse-down method informs the run time engine immediately of this event and the current mouse cursor coordinates. If the viewer releases the mouse at 259, the mouse-up method informs the run time engine immediately of this event. Either way, if the original "mouse down" event had occurred inside one or more bounding rectangles of any text button or image object(s) at 258, then appropriate two-dimensional status variables are set and the draw system is called. The "mouse up" event will cause the run time engine to reset those appropriate two-dimensional status variables, and then call the draw system. If the mouse was inside of the bounding rectangles of any image and/or text button object that had a defined text button, image and/or video pop-up object (See FIG. 16) with the freeze attribute, the appropriate two-dimensional status variables are set so that the draw system will not erase these pop-up objects when the mouse moves outside the bounding rectangle(s) of the parent text button or image objects. The draw system interprets the relevant two-dimensional variables for the existing text button and image object(s) on the current web page in its draw loop described above with reference to FIG. 16, and draws the correct backgrounds, text strings, images, and 3D frames based on the state of each object, as just set by the "mouseDown" computational two-dimensional loop.

If the 3D frame had been previously defined (See FIG. 16) to have the "live" setting, then the 3D effect is changed from a "raised" effect to a "depressed" effect. If the location of any of these text button and image objects or child pop-up objects is changing over time because of time line, animation, or transformation threads, the draw system, as described in association with FIG. 33 and FIG. 34, is aware of these dynamics, and redraws the screen as these real time events occur, but does not recognize any new "mouse down" or "mouse up" Events. In one implementation, only "mouse over" events are recognized dynamically by the draw system.

If any sound or video events were defined (See FIG. 16) for any text button or image objects, then the run time engine plays those sound and/or video files or channels as defined. As multiple objects can be defined that each have associated sound (and even video) files, and these objects can be overlaid on each other, either completely or partially, very interesting synchronized multiple sound tracks can be constructed, in which certain designated sounds are played, or stopped, based purely on the user pressing a mouse button.

If a mouse down event as described previously in association with FIG. 16 was defined for one or more effected text button or image objects, only the object that was drawn last will have its event processed. If the event was to go to a different internal web page, then the run time engine sets the web page counter described in association with FIG. 31 for the "current" page to be one less than that of the desired web page. The current web page's object time lines, if any, complete normally, and the desired web page is then executed. If the "mouse down" event was to go to an external web page in a different window, then the run time engine creates a new

64

browser window. In JAVA this can be accomplished with the "getAppletContext( ).showDocument(theURL, "\_blank")" method, where "theURL" is the URL address for the external web page. The run time engine, however, continues executing.

If the mouse down event was to go to an external web page in the same browser window, then the run time engine terminates by turning control over to the designated external web page. In JAVA this can be accomplished with the "getAppletContext( ).showDocument(theURL)" method, where "theURL" is the URL address for the external web page.

If the viewer presses a mouse button while inside the bounding rectangle for a paragraph hot link at 260, then appropriate four-dimensional status variables (web page by paragraph number, by paragraph line number, by paragraph line segment number) are set. If the hot link setting at FIG. 16 was to go to a different internal web page, then the run time engine sets the web page counter described previously in association with FIG. 31 for the "current" page to be one less than that of the desired web page. The current web page's object time lines, if any, complete normally, and the desired web page is then executed. If the hot link setting was to go to an external web page in a different window, then the run time engine creates a new browser window and loads the external web page. The run time engine, however, continues executing. If the hot link setting was to go to an external web page in the same browser window, then the run time engine terminates by turning control over to the designated external web page. This completes the detailed description of the run time process at 261.

As is obvious from the descriptions of the various features and processes described above, it will be apparent to one skilled in the art that variations in form and detail may be made in the preferred implementation and methods without varying from the spirit and scope of the invention as defined in the claims or as interpreted under the doctrine of equivalents. It should also be clear that terms such as "browser", "mouse", "server", "web", etc., while adequate to describe the current state of interactive systems such as the World Wide Web, may evolve into new and more powerful entities. This evolution of terminology and technology is independent of the preferred implementation and methods of the invention as defined in the claims or as interpreted under the doctrine of equivalents. The preferred implementation and methods are thus provided for purposes of explanation and illustration, but not limitation.

I claim:

1. A system for assembling a web site comprising:

a server comprising a build engine configured to:

accept user input to create a web site, the web site comprising a plurality of web pages, each web page comprising a plurality of objects,

accept user input to associate a style with objects of the plurality of web pages, wherein each web page comprises at least one button object or at least one image object, and wherein the at least one button object or at least one image object is associated with a style that includes values defining transformations and time lines for the at least one button object or at least one image object; and wherein each web page is defined entirely by each of the plurality of objects comprising that web page and the style associated with the object, produce a database with a multidimensional array comprising the objects that comprise the web site including data defining, for each object, the object style, an object number, and an indication of the web page that each object is part of, and

US 7,594,168 B2

**65**

provide the database to a server accessible to web browser;  
 wherein the database is produced such that a web browser with access to a runtime engine is configured to generate the web-site from the objects and style data extracted from the provided database.

2. The system of claim 1,

wherein one of said plurality of objects is a child, and

wherein the build engine is configured to accept user input to associate a style with child button and child image objects.

**66**

3. The system of claim 2, wherein at least one of said styles includes values defining time lines for child button and child image objects.

5 4. The system of claim 1, wherein at least one of said styles includes settings for multiple object states.

5. The system of claim 1, further including file size reduction means for reducing the total size of files generated by said build engine to a size of between 12K and 50K.

10 6. The system of claim 1, where said data is stored as one or more of a Boolean an integer, a string, a floating point variables, or a URL.

\* \* \* \* \*



US009063755B2

(12) **United States Patent**  
**Rempell et al.**

(10) **Patent No.:** **US 9,063,755 B2**  
(45) **Date of Patent:** **Jun. 23, 2015**

(54) **SYSTEMS AND METHODS FOR  
PRESENTING INFORMATION ON MOBILE  
DEVICES**

(75) Inventors: **Steven H. Rempell**, Novato, CA (US);  
**David Chrobak**, Clayton, CA (US); **Ken  
Brown**, San Martin, CA (US)

(73) Assignee: **Express Mobile, inc.**, Novato, CA (US)

(\*) Notice: Subject to any disclaimer, the term of this  
patent is extended or adjusted under 35  
U.S.C. 154(b) by 154 days.

(21) Appl. No.: **12/936,395**

(22) PCT Filed: **Apr. 6, 2009**

(86) PCT No.: **PCT/US2009/039695**

§ 371 (c)(1),  
(2), (4) Date: **Nov. 3, 2010**

(87) PCT Pub. No.: **WO2009/126591**

PCT Pub. Date: **Oct. 15, 2009**

(65) **Prior Publication Data**

US 2011/0107227 A1 May 5, 2011

**Related U.S. Application Data**

(60) Provisional application No. 61/166,651, filed on Apr. 3, 2009, provisional application No. 61/113,471, filed on Nov. 11, 2008, provisional application No. 61/123,438, filed on Apr. 7, 2008.

(51) **Int. Cl.**  
**G06F 3/048** (2013.01)  
**G06F 9/44** (2006.01)  
**G06F 9/45** (2006.01)

(52) **U.S. Cl.**  
CPC ..... **G06F 9/4443** (2013.01)

(58) **Field of Classification Search**  
CPC ..... G06F 3/048  
USPC ..... 715/738  
See application file for complete search history.

(56) **References Cited**

**U.S. PATENT DOCUMENTS**

2004/0055017	A1 *	3/2004	Delpuch et al.	725/110
2004/0163020	A1 *	8/2004	Sidman	714/100
2005/0149935	A1 *	7/2005	Benedetti	718/102
2005/0273705	A1 *	12/2005	McCain	715/513
2006/0063518	A1 *	3/2006	Paddon et al.	455/418

**OTHER PUBLICATIONS**

Stina Nylander et al. "The Ubiquitous Interactor-Device Independent Access to Mobile Services" (Computer-Aided Design for User Interfaces IV, Proceedings of the Fifth International Conference on Computer-Aided Design of User Interfaces CADUI'2004, Jan. 2004, pp. 271-282).\*

\* cited by examiner

*Primary Examiner* — Jennifer To

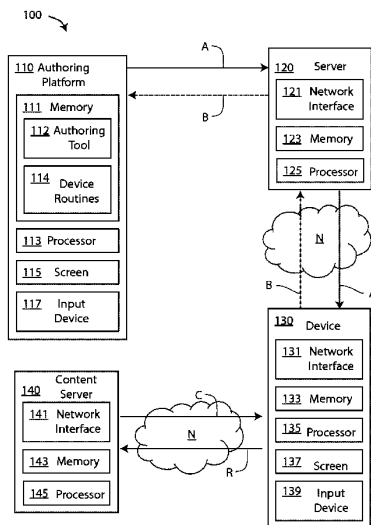
*Assistant Examiner* — Xuyang Xia

(74) *Attorney, Agent, or Firm* — Steven R. Vosen

(57) **ABSTRACT**

Embodiments of a system and method are described for generating and distributing programming to mobile devices over a network. Devices are provided with Players specific to each device and Applications that are device independent. Embodiments include a full-featured WYSIWYG authoring environment, including the ability to bind web components to objects.

**28 Claims, 18 Drawing Sheets**

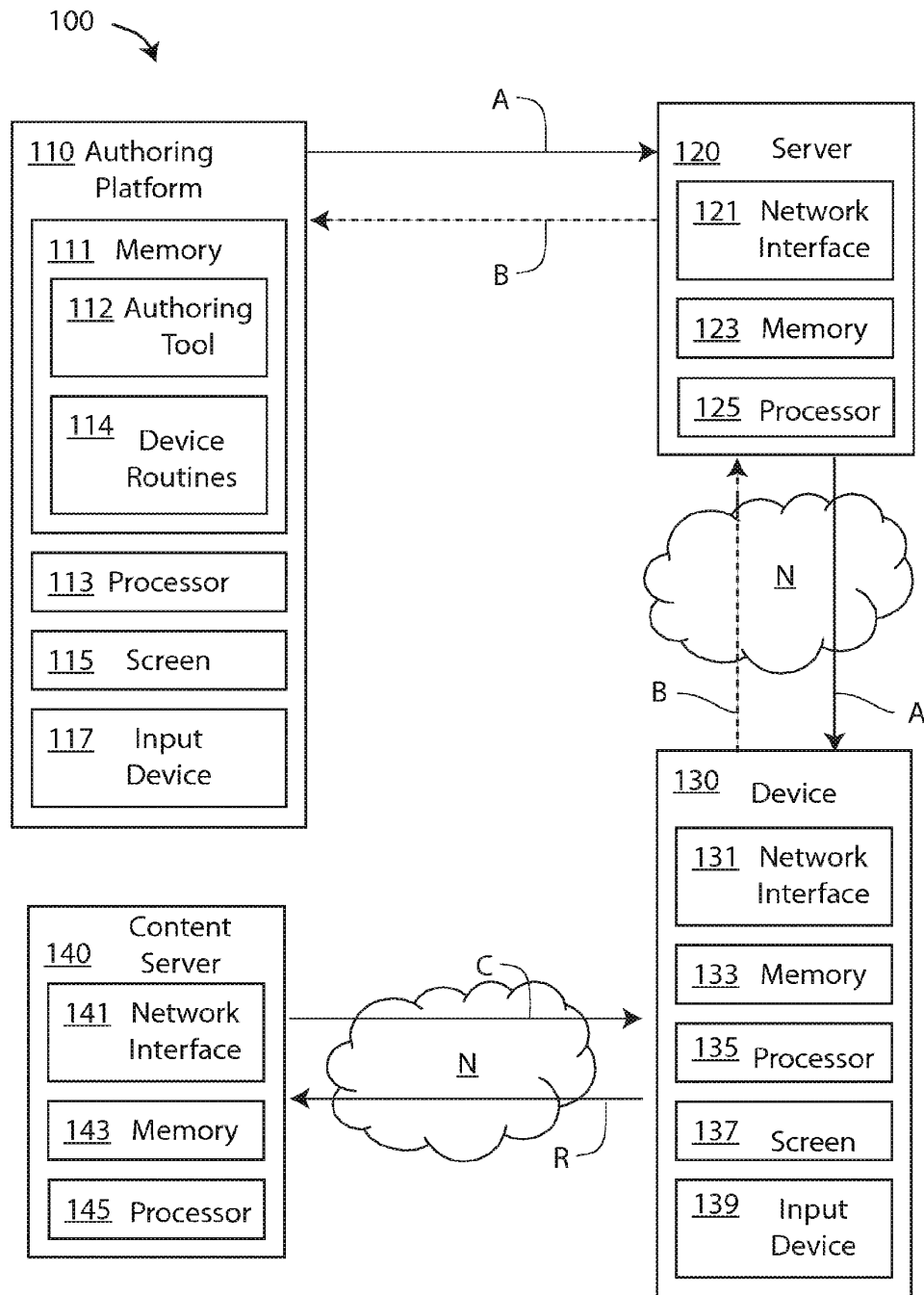


**U.S. Patent**

Jun. 23, 2015

Sheet 1 of 18

**US 9,063,755 B2**



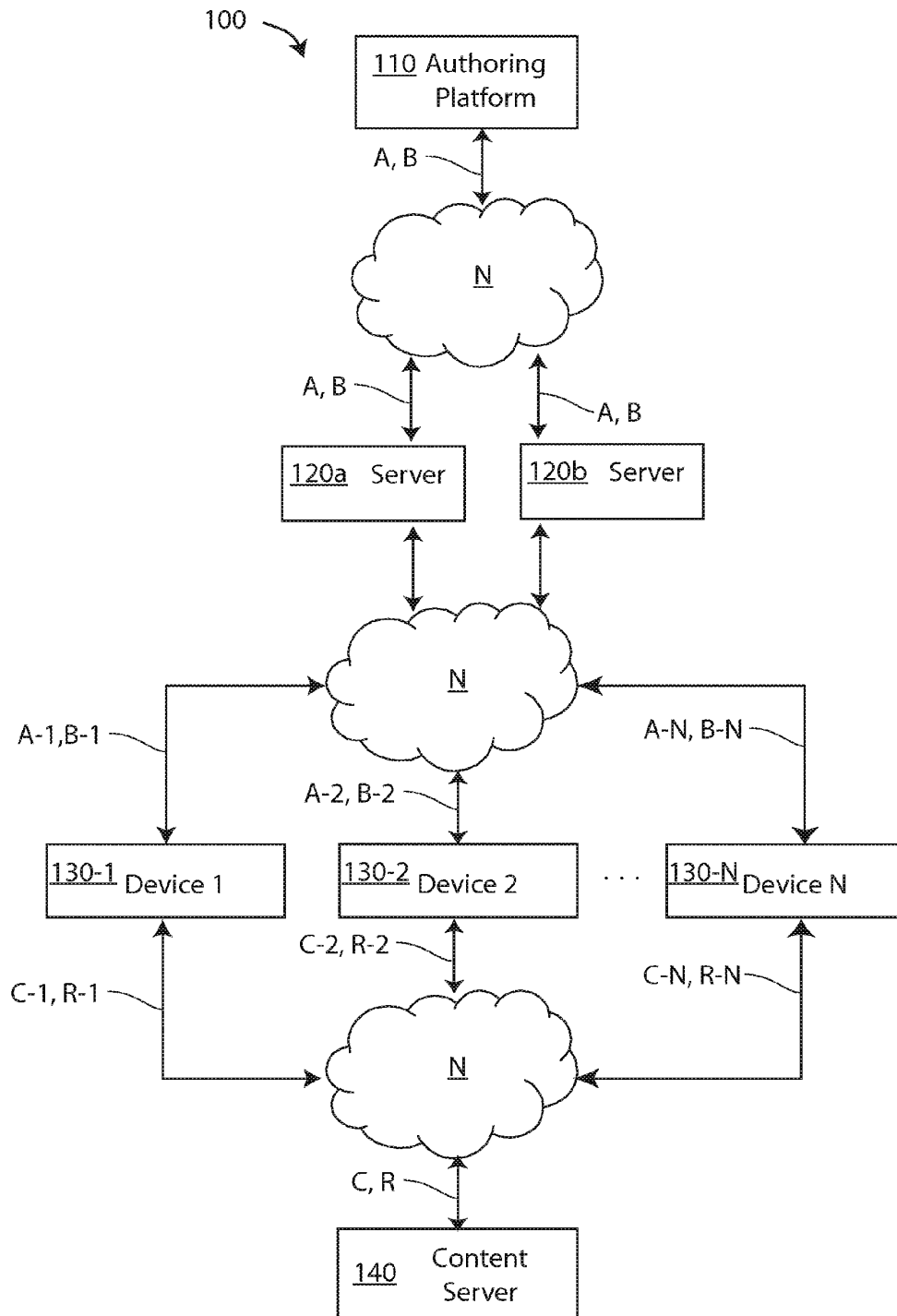
**FIG. 1A**

**U.S. Patent**

**Jun. 23, 2015**

**Sheet 2 of 18**

**US 9,063,755 B2**



**FIG. 1B**

U.S. Patent

Jun. 23, 2015

Sheet 3 of 18

US 9,063,755 B2

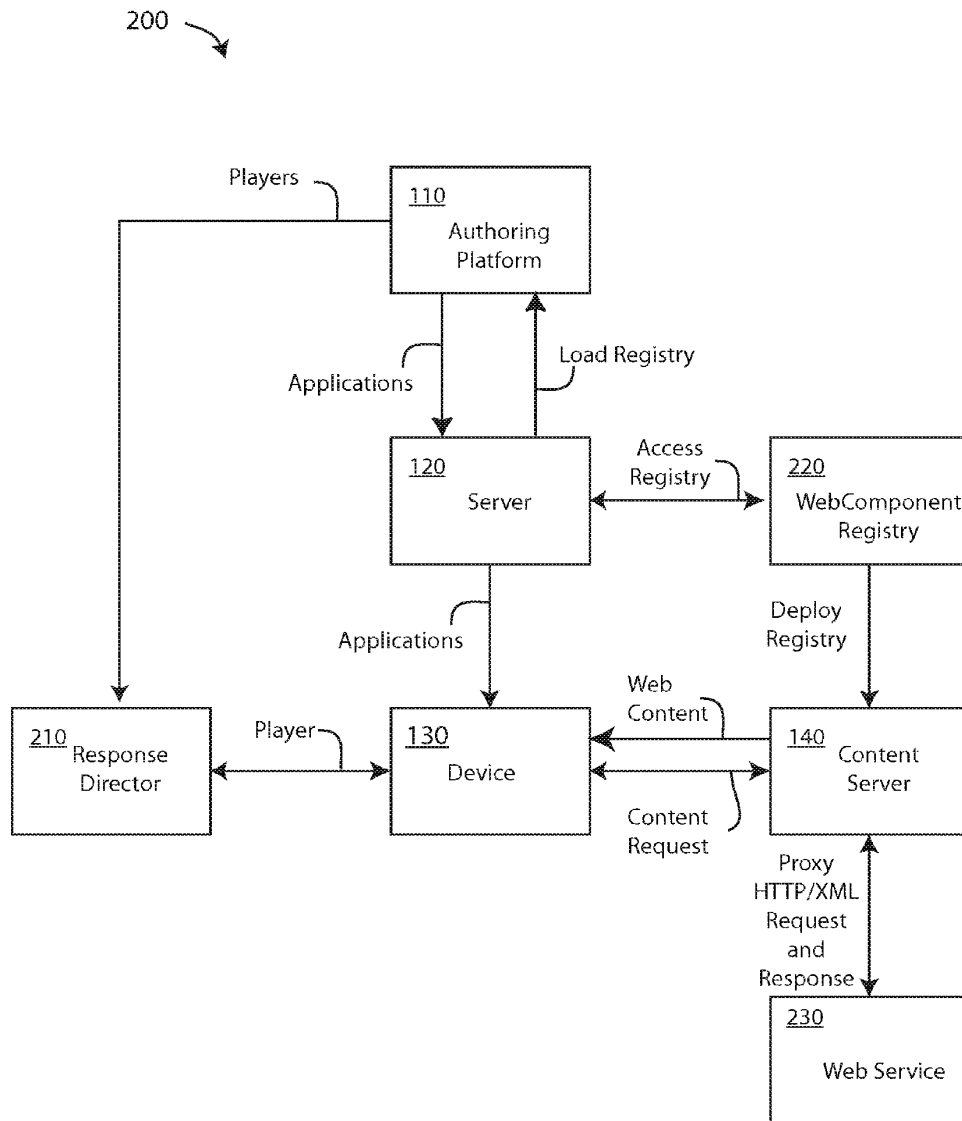


FIG. 2A

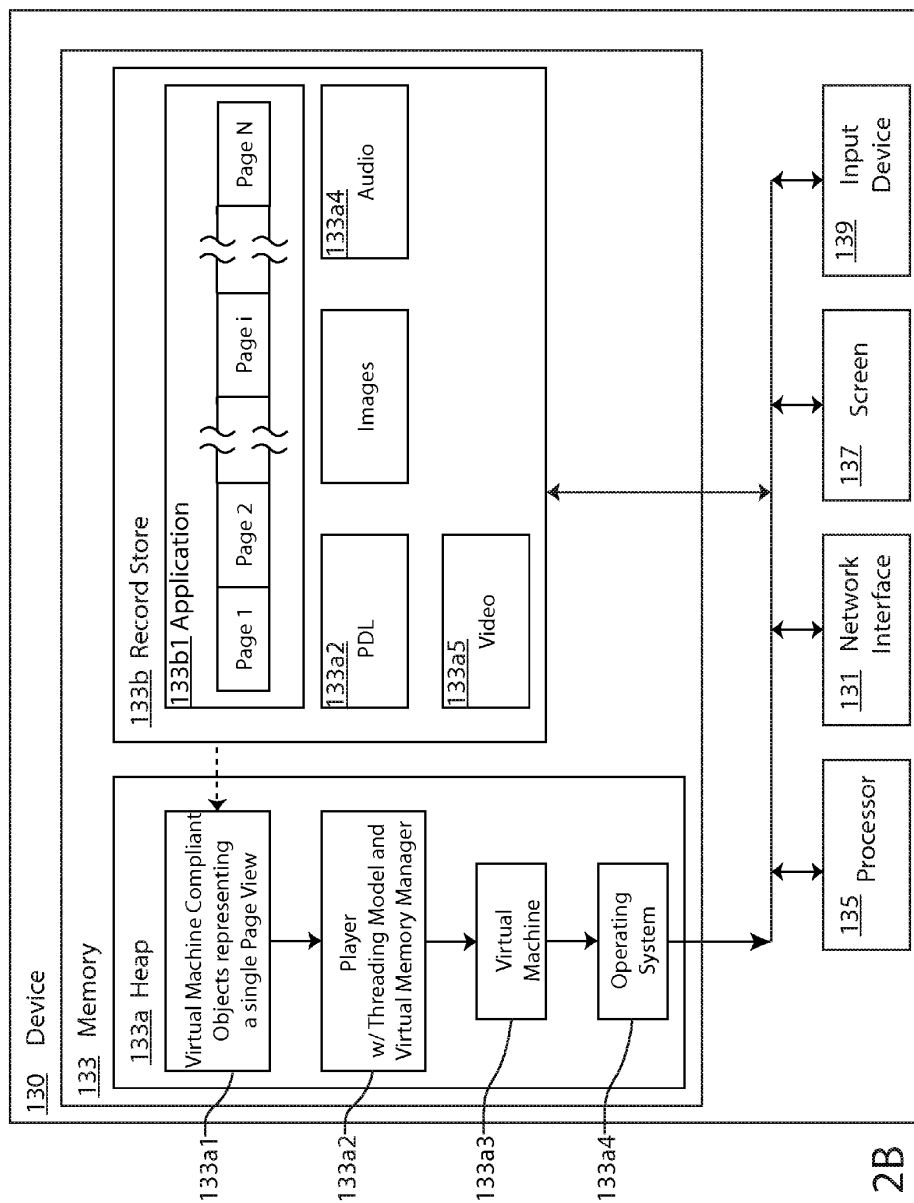
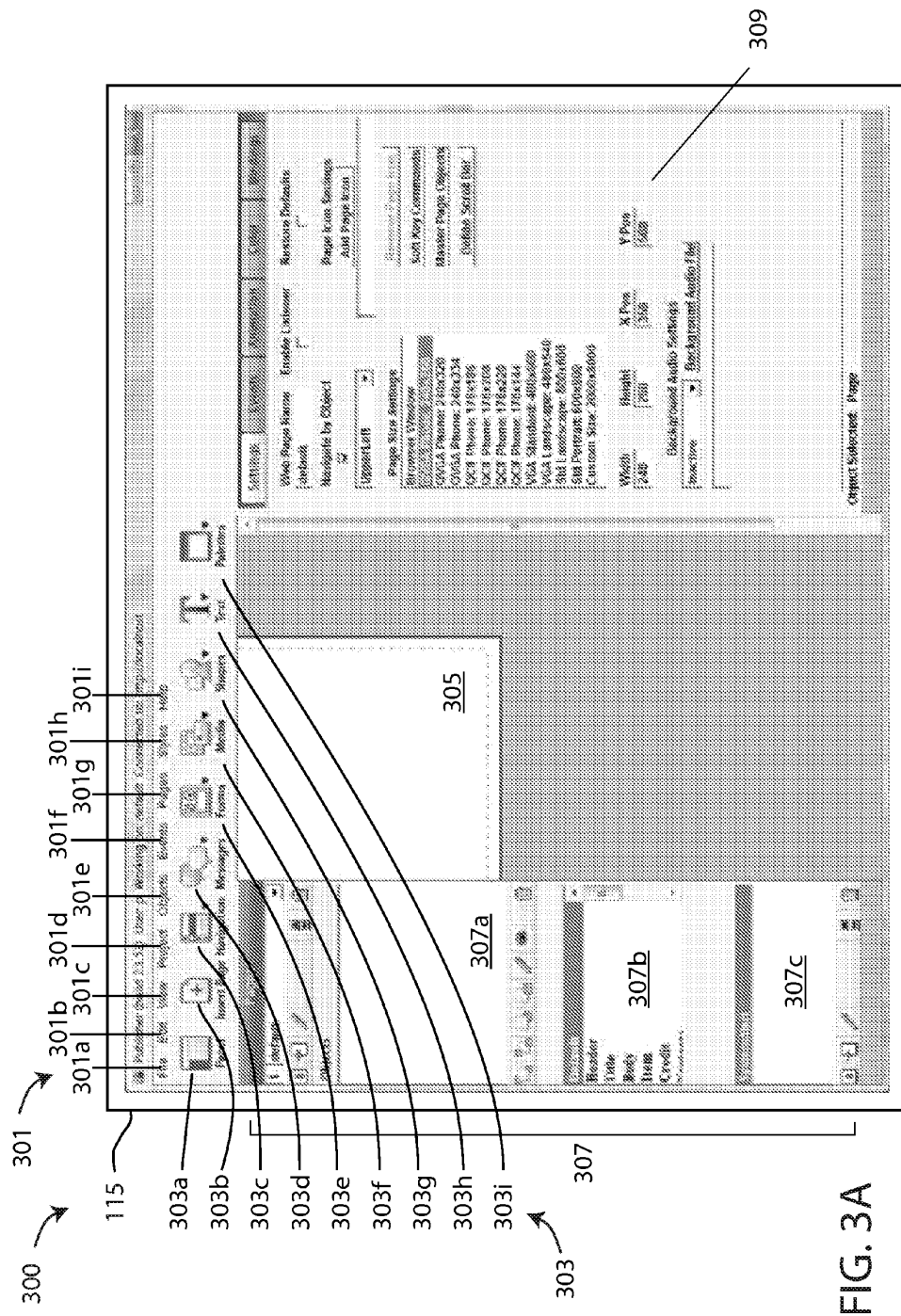
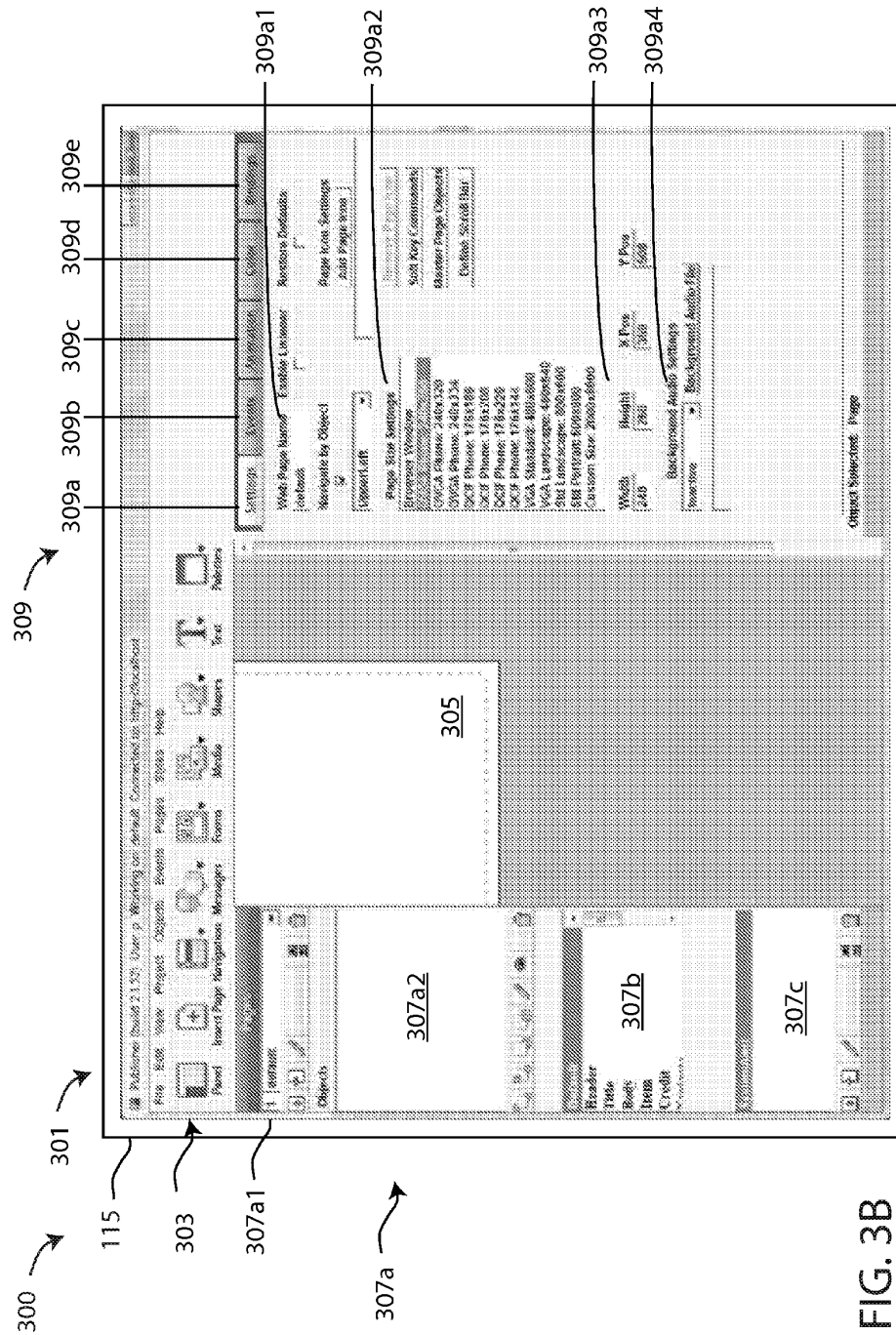


FIG. 2B







U.S. Patent

Jun. 23, 2015

Sheet 7 of 18

US 9,063,755 B2

+

309b

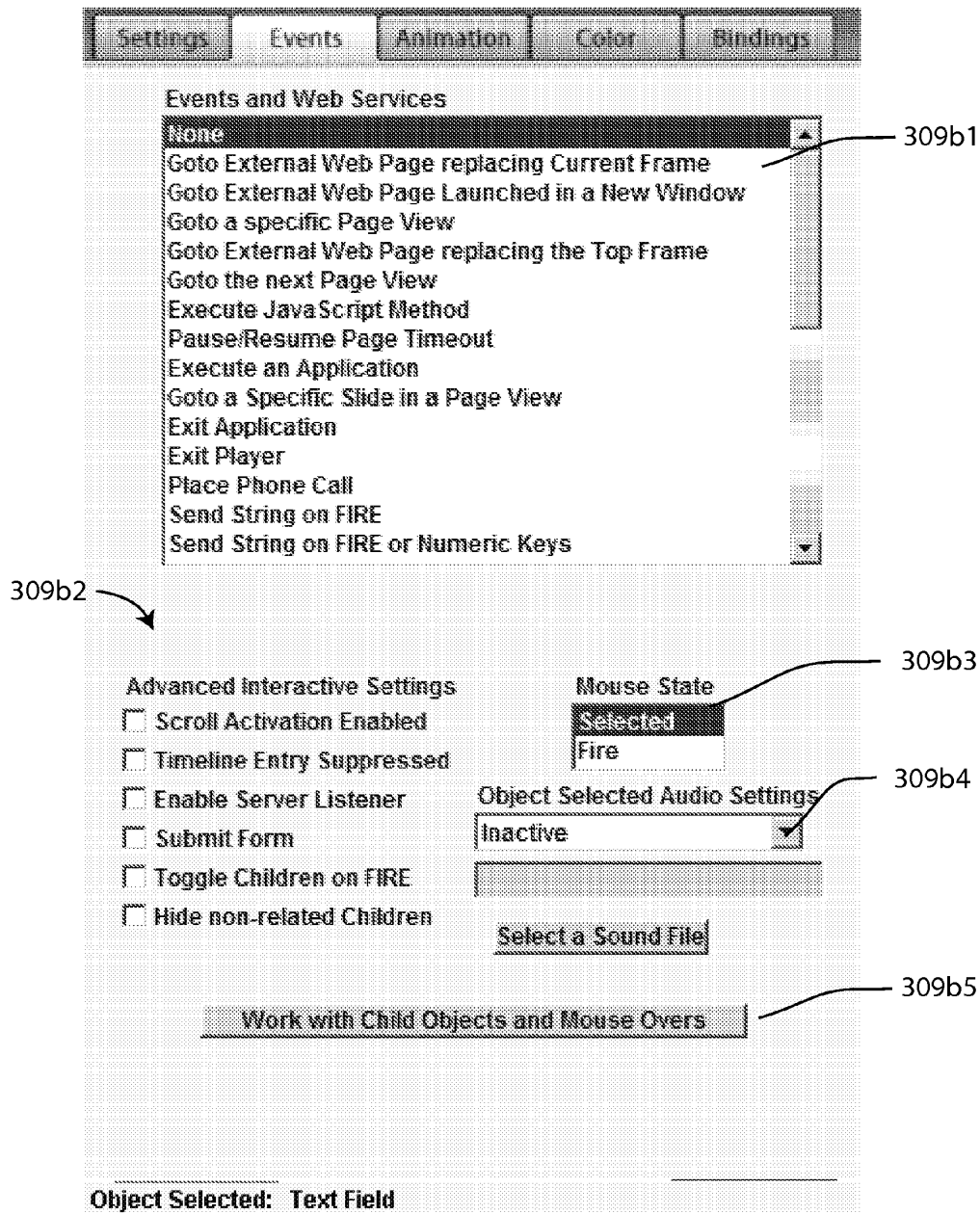


FIG. 3C

U.S. Patent

Jun. 23, 2015

Sheet 8 of 18

US 9,063,755 B2

309c

Settings Events Animation Color Bindings

**Activate** Object Entry Timeline Specifications

Timeline ☐ Delay (Sec) 0 .0 Direction None Movement None Duration (Sec) 2 .0 Frames 10

Specifications for this Object's Entry Animation Audio Track

Inactive Entry Audio File

**Object Animation Specifications**

Delay (Sec)	Direction	Movement	Duration (Sec)	Frames
0	None	None	0	1
1	Scroll Left	Fade	1	2
2	Scroll Right	Fade In	2	3
3	Custom	Fade Out	3	4
4	Multi-Point		4	5
5	Seek Cursor		5	6
6	Attach		6	7
7	Deposit		7	8
8	Send Home		8	9
9	Carom N		9	10
10	Carom NE		10	11

Pathname for this Object's Animation Audio Track

Inactive Main Audio File

Animation Cycles 1 Custom Zoom % 0 Avoid Cursor ☒ Dampen Anim. ☐

**Object Exit Timeline Specifications**

Activate ☐ Delay (Sec) 0 .0 Direction None Movement None Duration (Sec) 2 .0 Frames 10

Specifications for this Object's Exit Animation Audio Track

Inactive Exit Audio File

FIG. 3D

U.S. Patent

Jun. 23, 2015

Sheet 9 of 18

US 9,063,755 B2

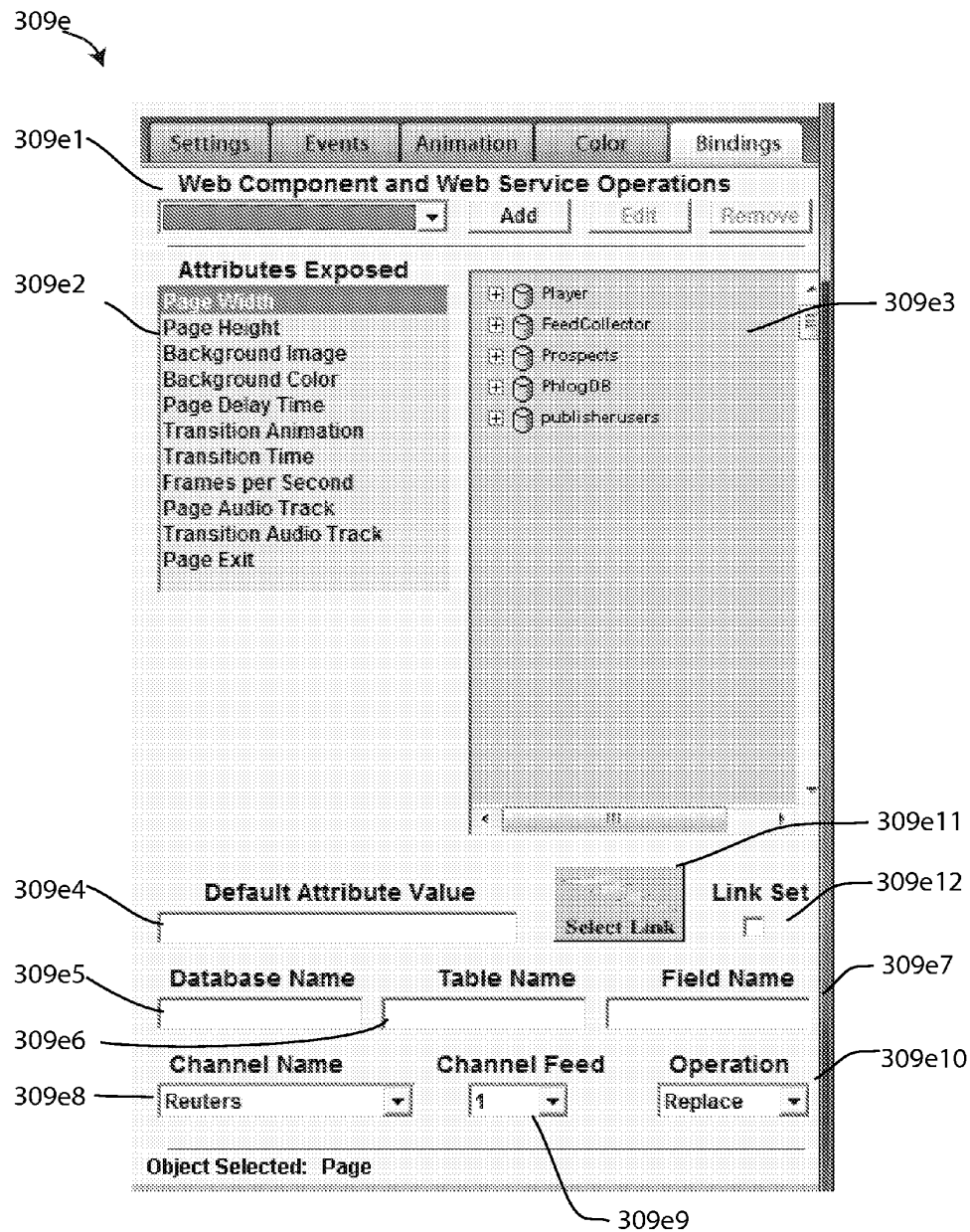


FIG. 3E

U.S. Patent

Jun. 23, 2015

Sheet 10 of 18

US 9,063,755 B2

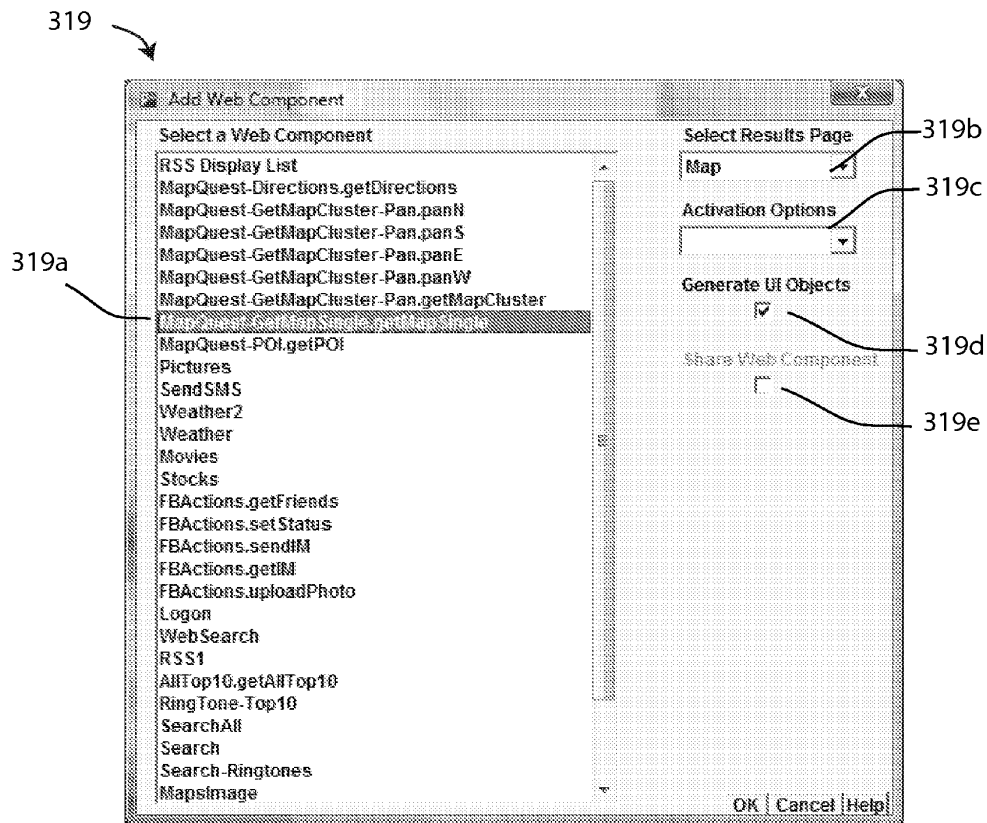


FIG. 3F

U.S. Patent

Jun. 23, 2015

Sheet 11 of 18

US 9,063,755 B2

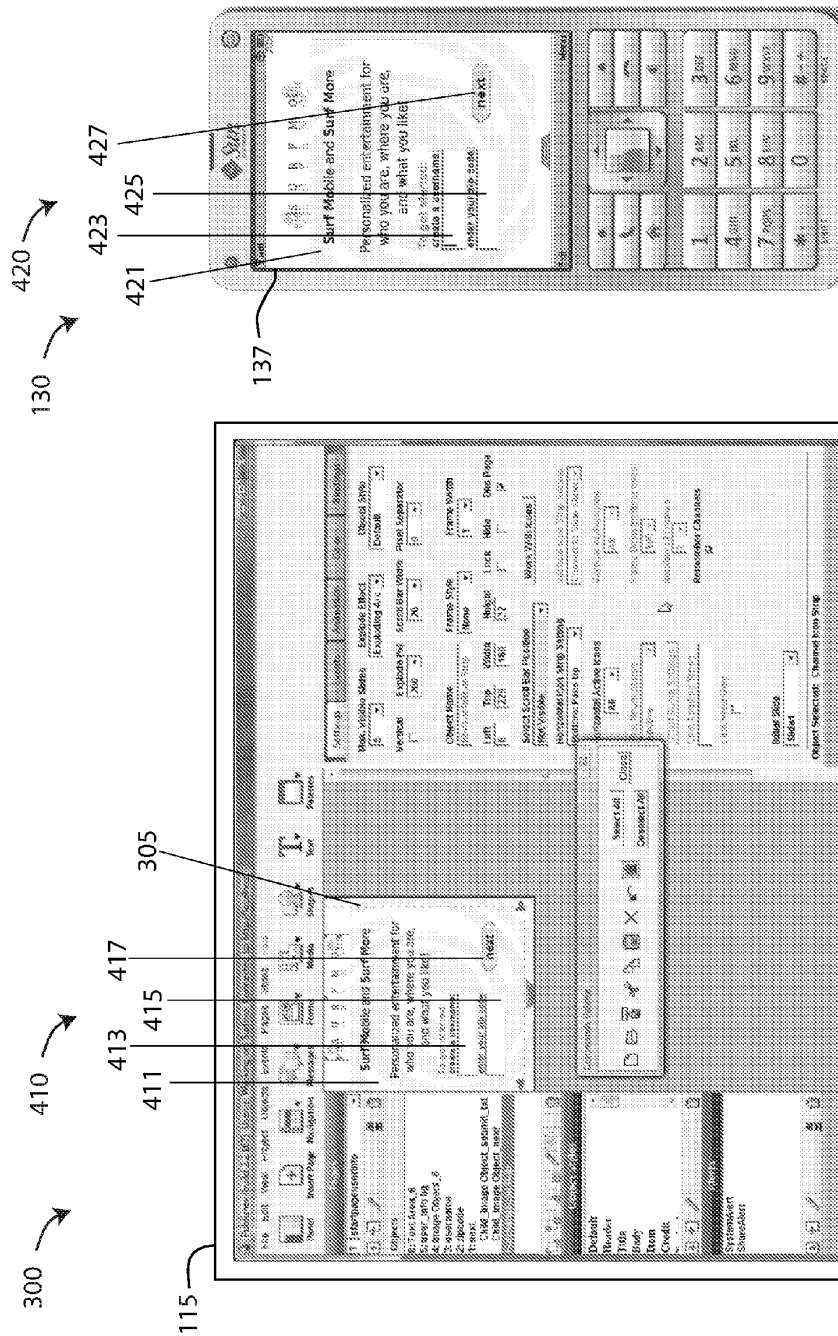


FIG. 4B

FIG. 4A

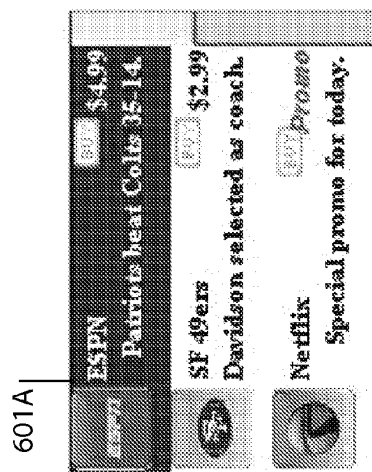


FIG. 6A

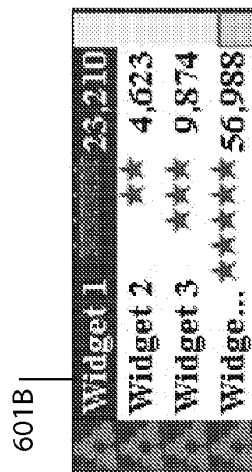


FIG. 6B

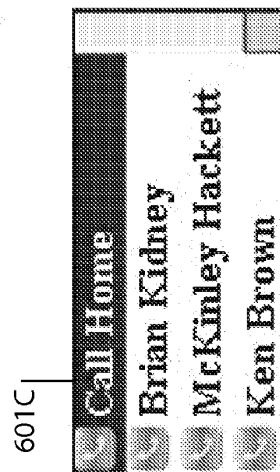


FIG. 6C

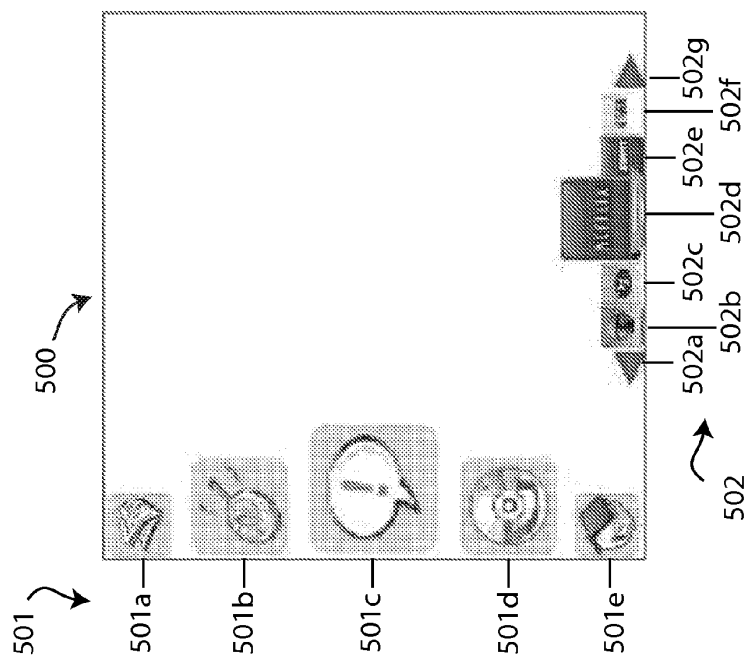


FIG. 5



U.S. Patent

Jun. 23, 2015

Sheet 13 of 18

US 9,063,755 B2

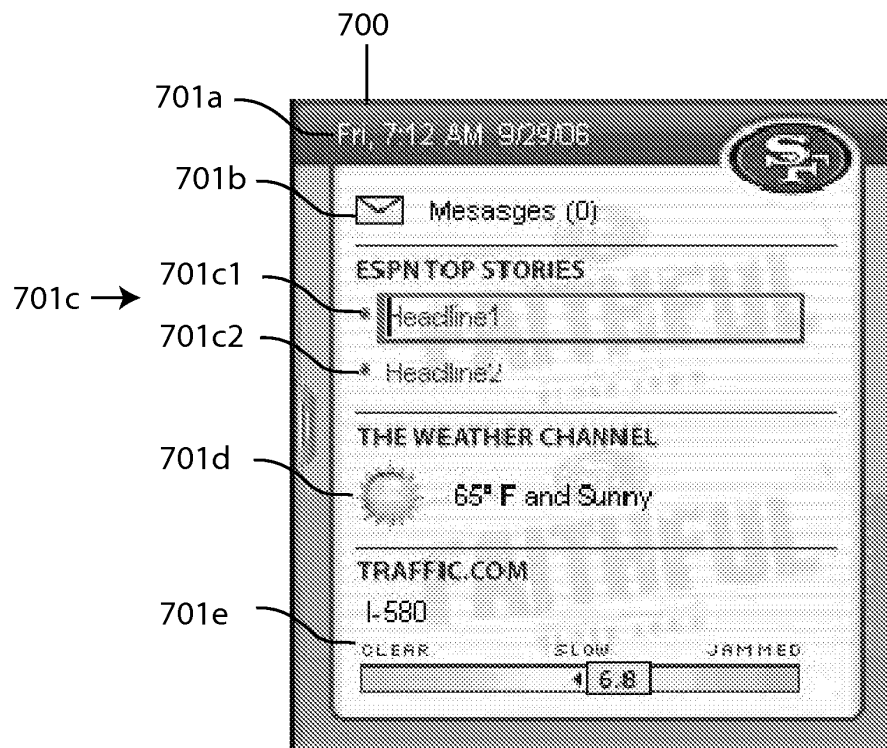


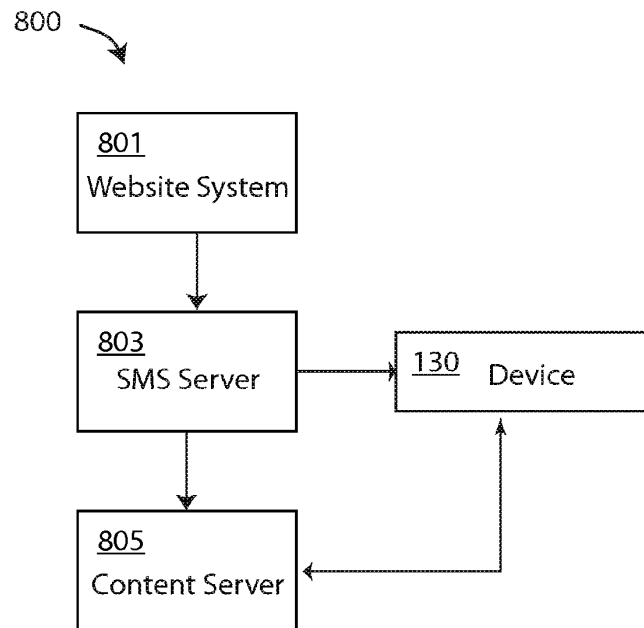
FIG. 7

**U.S. Patent**

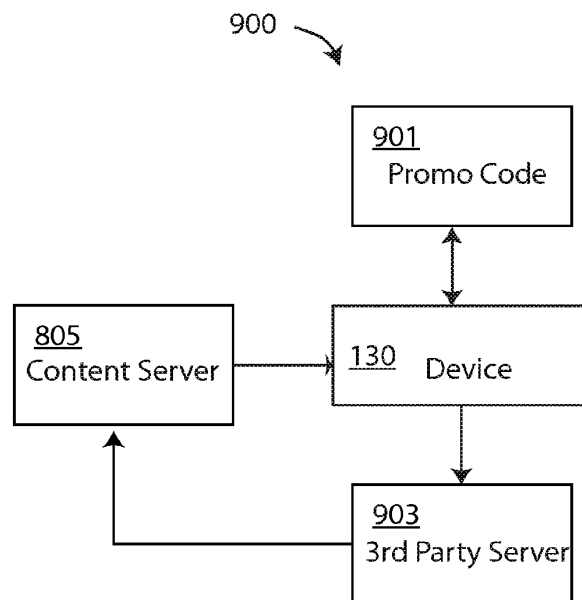
**Jun. 23, 2015**

**Sheet 14 of 18**

**US 9,063,755 B2**



**FIG. 8**



**FIG. 9**

U.S. Patent

Jun. 23, 2015

Sheet 15 of 18

US 9,063,755 B2

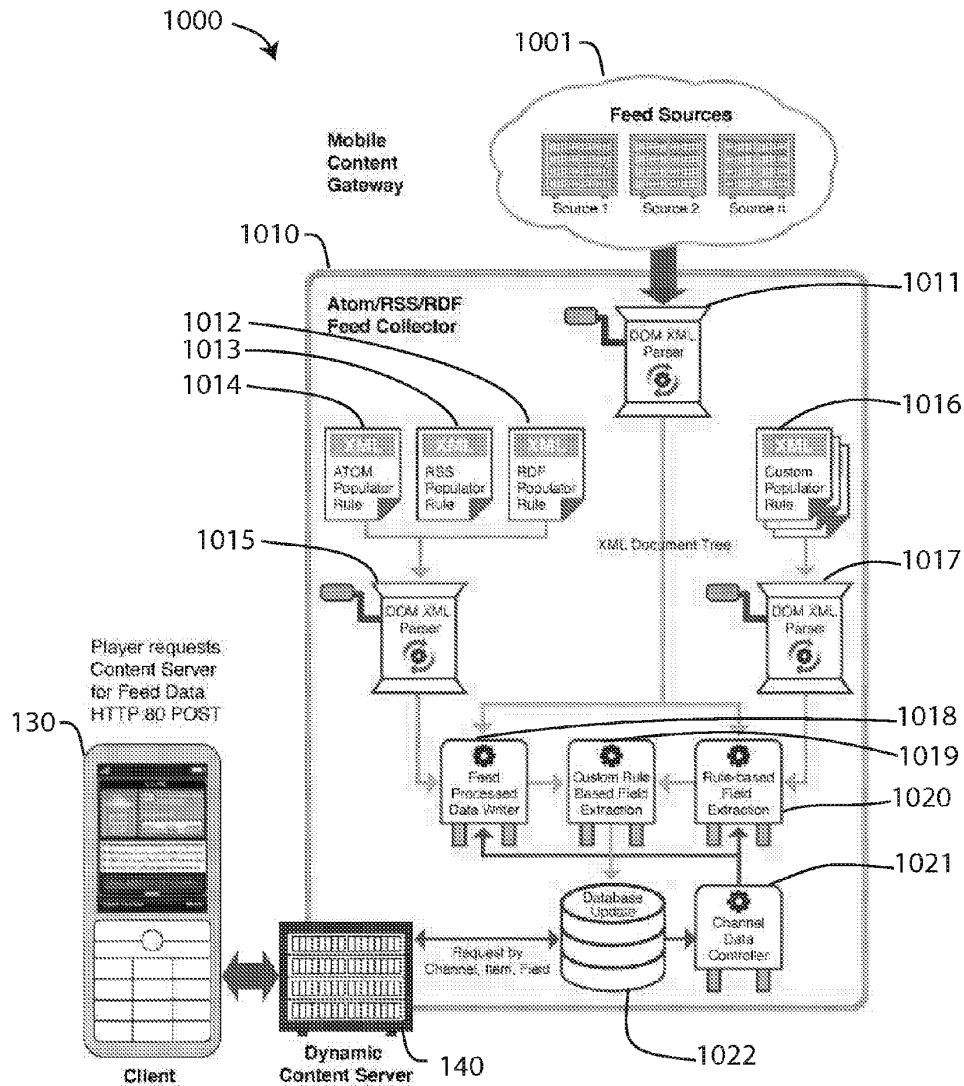


FIG. 10

U.S. Patent

Jun. 23, 2015

Sheet 16 of 18

US 9,063,755 B2

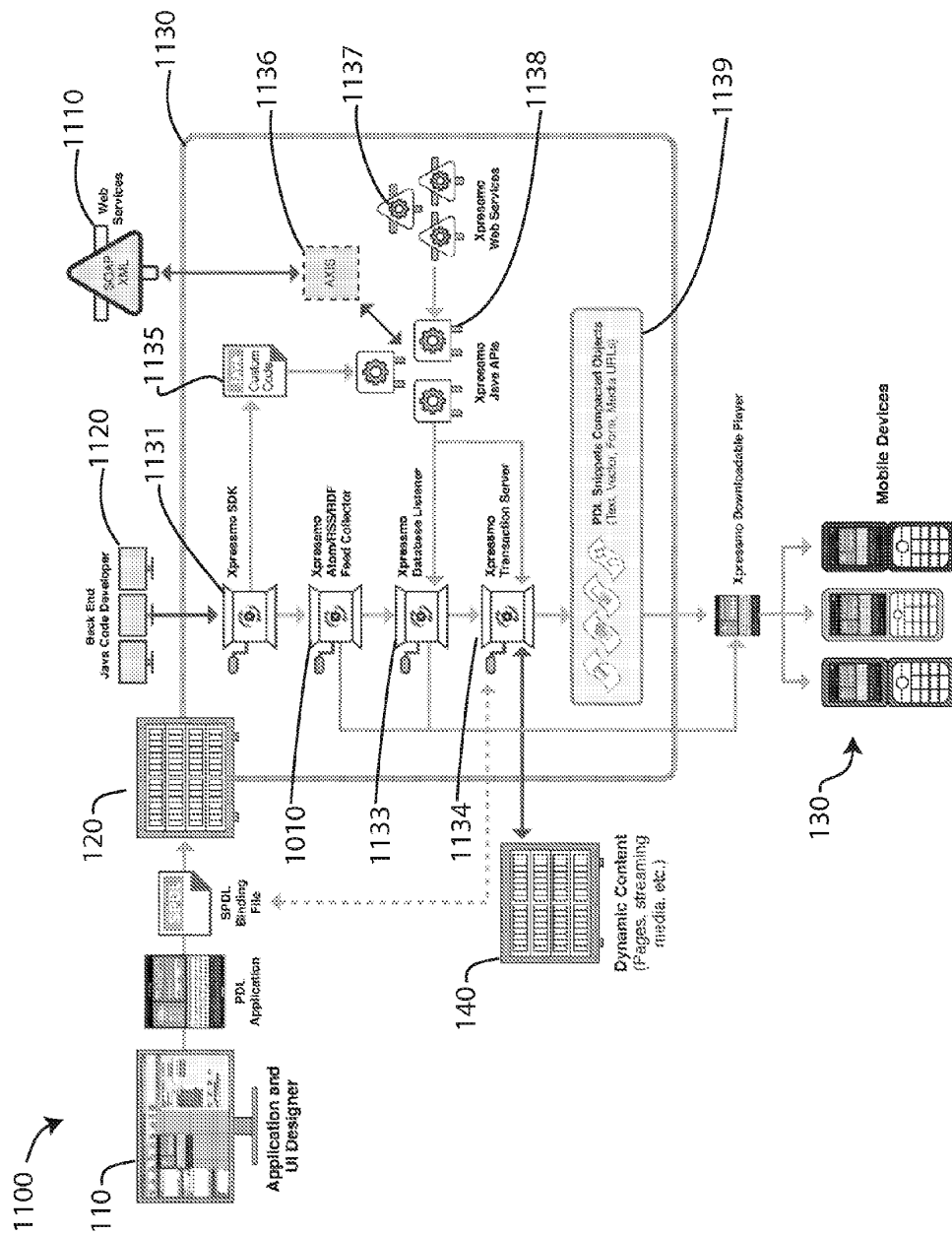


FIG. 11

U.S. Patent

Jun. 23, 2015

Sheet 17 of 18

US 9,063,755 B2

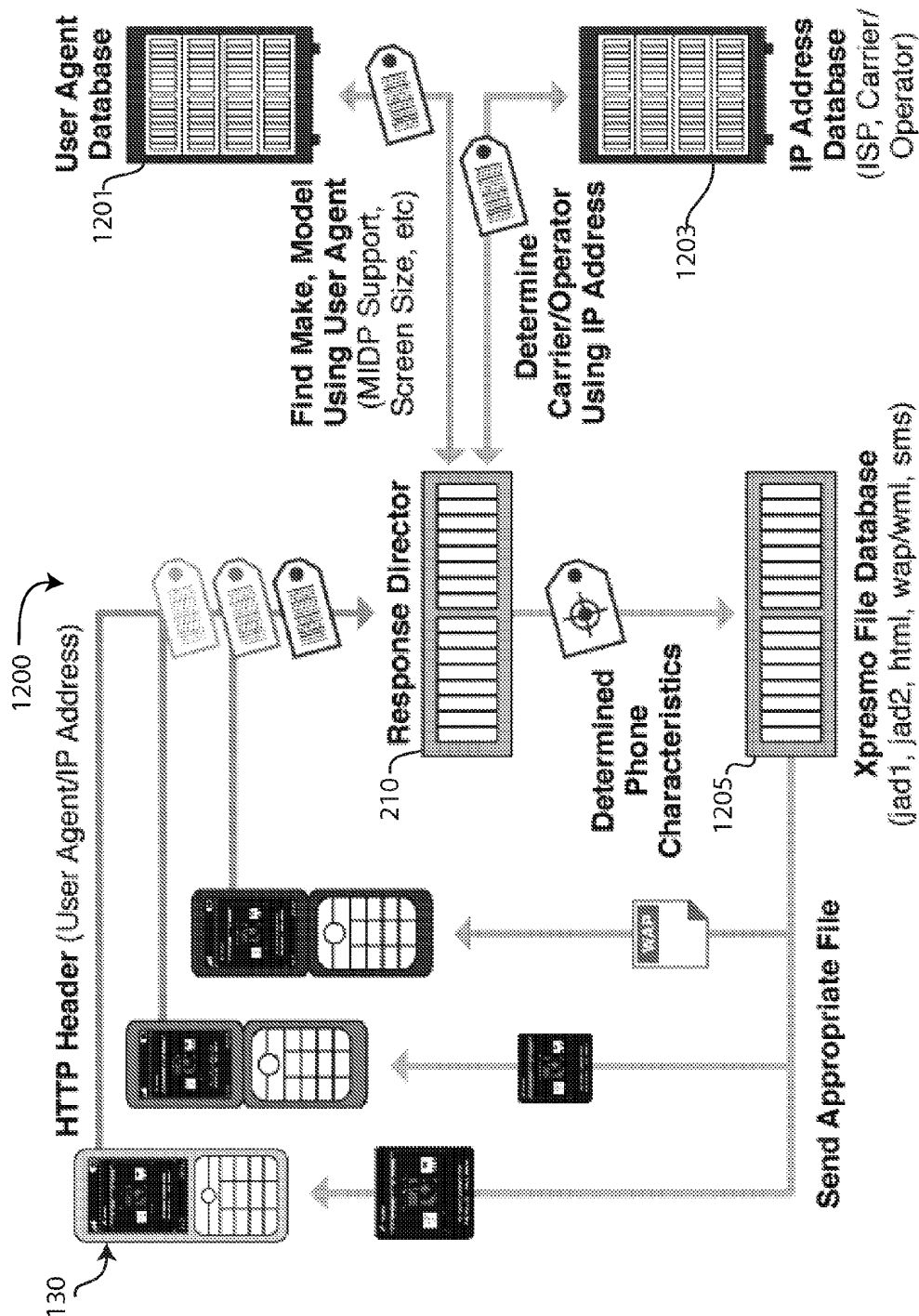


FIG. 12

U.S. Patent

Jun. 23, 2015

Sheet 18 of 18

US 9,063,755 B2

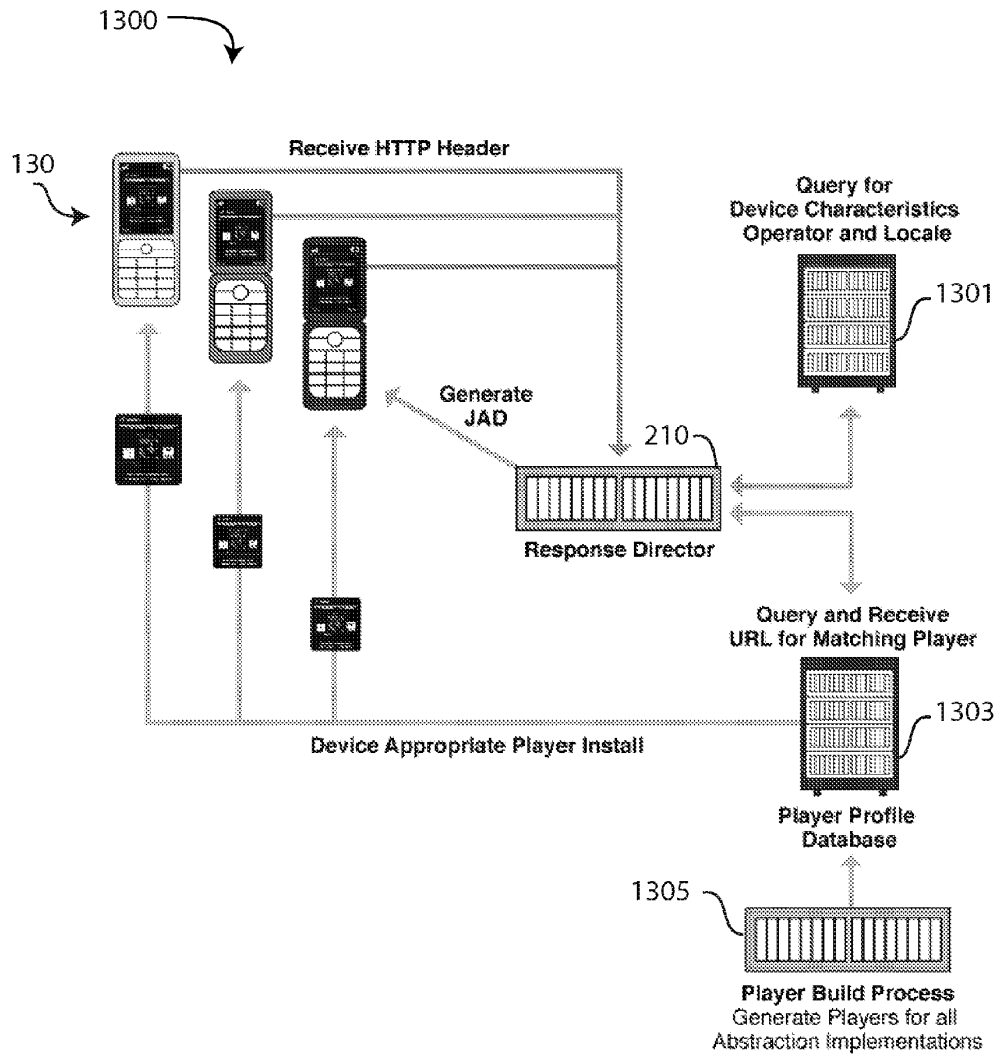


FIG. 13

US 9,063,755 B2

1

## SYSTEMS AND METHODS FOR PRESENTING INFORMATION ON MOBILE DEVICES

### TECHNICAL FIELD

The present invention generally relates to providing software for mobile devices, and more particularly to a method and system for authoring Applications for devices.

### BACKGROUND ART

Internet-connected mobile devices are becoming ever more popular. While these devices provide portability to the Internet, they generally do not have the capabilities of non-mobile devices including computing, input and output capabilities.

In addition, the mobility of the user while using such devices provides challenges and opportunities for the use of the Internet. Further, unlike non-mobile devices, there are a large number of types of devices and they tend to have a shorter lifetime in the marketplace. The programming of the myriad of mobile devices is a time-consuming and expensive proposition, thus limiting the ability of service providers to update the capabilities of mobile devices.

Thus there is a need in the art for a method and apparatus that permits for the efficient programming of mobile devices. Such a method and apparatus should be easy to use and provide output for a variety of devices.

### DISCLOSURE OF INVENTION

In certain embodiments, a system is provided to generate code to provide content on a display of a platform. The system includes a database of web services obtainable over a network and an authoring tool. The authoring tool is configured to define an object for presentation on the display, select a component of a web service included in said database, associate said object with said selected component, and produce code that, when executed on the platform, provides said selected component on the display of the platform.

In certain other embodiments, a method is provided for providing information to platforms on a network. The method includes accepting a first code over the network, where said first code is platform-dependent; providing a second code over the network, where said second code is platform-independent; and executing said first code and said second code on the platform to provide web components obtained over the network.

In certain embodiments, a method for displaying content on a platform utilizing a database of web services obtainable over a network is provided. The method includes: defining an object for presentation on the display; selecting a component of a web service included in said database; associating said object with said selected component; and producing code that, when executed on the platform, provides said selected component on the display of the platform.

In one embodiment, one of the codes is a Player, which is a thin client architecture that operates in a language that manages resources efficiently, is extensible, supports a robust application model, and has no device specific dependencies. In another embodiment, Player P is light weight and extends the operating system and/or virtual machine of the device to: Manage all applications and application upgrades, and resolve device, operating system, VM and language fragmentation.

2

In another embodiment, one of the codes is an Application that is a device independent code that interpreted by the Player.

These features together with the various ancillary provisions and features which will become apparent to those skilled in the art from the following detailed description, are attained by the system and method of the present invention, preferred embodiments thereof being shown with reference to the accompanying drawings, by way of example only, wherein:

### BRIEF DESCRIPTION OF DRAWINGS

FIG. 1A is an illustrative schematic of one embodiment of a system including an authoring platform and a server for providing programming instructions to a device over a network;

FIG. 1B is schematic of an alternative embodiment system for providing programming instructions to device over a network;

FIG. 2A is a schematic of an embodiment of system illustrating the communications between different system components;

FIG. 2B is a schematic of one embodiment of a device illustrating an embodiment of the programming generated by authoring platform;

FIGS. 3A and 3B illustrate one embodiment of a publisher interface as it appears, for example and without limitation, on a screen while executing an authoring tool;

FIG. 3C illustrates an embodiment of the Events Tab'

FIG. 3D illustrates one embodiment of an Animation Tab;

FIG. 3E illustrates one embodiment of Bindings Tab;

FIG. 3F illustrates one embodiment of a pop-up menu for adding web components;

FIG. 4A shows a publisher interface having a layout on a canvas; and FIG. 4B shows a device having the resulting layout on a device screen;

FIG. 5 shows a display of launch strips;

FIG. 6A is a display of a Channel Selection List;

FIG. 6B is a display of a Widget Selection List;

FIG. 6C is a display of a Phone List;

FIG. 7 shows a display of a mash-up;

FIG. 8 is a schematic of an embodiment of a push capable system;

FIG. 9 is a schematic of an alternative embodiment of a push capable system;

FIG. 10 is a schematic of one embodiment of a feed collector;

FIG. 11 is a schematic of an embodiment of a Mobile Content Gateway;

FIG. 12 is a schematic of one embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database; and

FIG. 13 is a schematic of another embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database.

Reference symbols are used in the Figures to indicate certain components, aspects or features shown therein, with reference symbols common to more than one Figure indicating like components, aspects or features shown therein.

### MODE(S) FOR CARRYING OUT THE INVENTION

FIG. 1A is an illustrative schematic of one embodiment of a system **100** including an authoring platform **110** and a server **120** for providing programming instructions to a

US 9,063,755 B2

3

device 130 over a network N. In one embodiment, device 130 is a wireless device, and network N includes wireless communication to the device. Alternatively, system 100 may provide access over network N to other information, data, or content, such as obtainable as a web service over the Internet. In general, a user of authoring platform 110 may produce programming instructions or files that may be transmitted over network N to operate device 130, including instructions or files that are sent to device 130 and/or server 120. The result of the authoring process is also referred to herein, and without limitation, as publishing an Application.

Embodiments include one or more databases that store information related to one or more devices 130 and/or the content provided to the devices. It is understood that such databases may reside on any computer or computer system on network N, and that, in particular, the location is not limited to any particular server, for example.

Device 130 may be, for example and without limitation, a cellular telephone or a portable digital assistant, includes a network interface 131, a memory 133, a processor 135, a screen 137, and an input device 139. Network interface 131 is used by device 130 to communication over a wireless network, such as a cellular telephone network, a WiFi network or a WiMax network, and then to other telephones through a public switched telephone network (PSTN) or to a satellite, or over the Internet. Memory 133 includes programming required to operate device 130 (such as an operating system or virtual machine instructions), and may include portions that store information or programming instructions obtained over network interface 131, or that are input by the user (such as telephone numbers or images from a device camera (not shown)). In one embodiment screen 137 is a touch screen, providing the functions of the screen and input device 139.

Authoring platform 110 includes a computer or computer system having a memory 111, a processor 113, a screen 115, and an input device 117. It is to be understood that memory 111, processor 113, screen 115, and input device 117 are configured such a program stored in the memory may be executed by the processor to accept input from the input device and display information on the screen. Further, the program stored in memory 111 may also instruct authoring platform 110 to provide programming or information, as indicated by the line labeled "A" and to receive information, as indicated by the line labeled "B."

Memory 111 is shown schematically as including a stored program referred to herein, and without limitation, as an authoring tool 112. In one embodiment, authoring tool 112 is a graphical system for designing the layout of features as a display that is to appear on screen 137. One example of authoring tool 112 is the CDER™ publishing platform (Express Mobile, Inc., Novato, Calif.).

In another embodiment, which is not meant to limit the scope of the present invention, device 130 may include an operating system having a platform that can interpret certain routines. Memory 111 may optionally include programming referred to herein, and without limitation, as routines 114 that are executable on device 130.

Routines 114 may include device-specific routines—that is, codes that are specific to the operating system, programming language, or platform of specific devices 130, and may include, but are not limited to, Java, Windows Mobile, Brew, Symbian OS, or Open Handset Alliance (OHA). Several examples and embodiments herein are described with reference to the use of Java. It is to be understood that the invention is not so limited, except as provided in the claims, and that one skilled in the art could provide Players for devices using routines provided on a platform. Thus as an example, routines

4

114 may include Java API's and an authoring tool System Development Kit (SDK) for specific devices 130.

Server 120 is a computer or computer system that includes a network interface 121, a memory 123, and a processor 125. It is to be understood that network interface 121, memory 123, and processor 125 are configured such that a program stored in the memory may be executed by the processor to: accept input and/or provide output to authoring platform 110; accept input and/or provide output through network interface 121 over network N to network interface 131; or store information from authoring platform 110 or from device 130 for transmission to another device or system at a later time.

In one embodiment, authoring platform 110 permits a user to design desired displays for screen 137 and actions of device 130. In other words, authoring platform 110 is used to program the operation of device 130. In another embodiment, authoring platform 110 allows a user to provide input for the design of one or more device displays and may further allow the user to save the designs as device specific Applications. The Applications may be stored in memory 123 and may then be sent, when requested by device 130 or when the device is otherwise accessible, over network N, through network interface 130 for storage in memory 133.

In an alternative embodiment, analytics information from devices 130 may be returned from device 130, through network N and server 120, back to authoring platform 110, as indicated by line B, for later analysis. Analytics information includes, but is not limited to, user demographics, time of day, and location. The type of analytic content is only limited by which listeners have been activated for which objects and for which pages. Analytic content may include, but is not limited to, player-side page view, player-side forms-based content, player-side user interactions, and player-side object status.

Content server 140 is a computer or computer system that includes a network interface 141, a memory 143, and a processor 145. It is to be understood that network interface 141, memory 143, and processor 145 are configured such that a stored program in the memory may be executed by the processor to accept requests R from device 130 and provide content C over a network, such as web server content the Internet, to device 130.

FIG. 1B is schematic of an alternative embodiment system 100 for providing programming instructions to device 130 over a network N that is generally similar to the system of FIG. 1A. The embodiment of FIG. 1B illustrates that system 100 may include multiple servers 120 and/or multiple devices 130.

In the embodiment of FIG. 1B, system 100 is shown as including two or more servers 120, shown illustratively and without limitation as servers 120a and 120b. Thus some of the programming or information between authoring platform 110 and one or more devices 130 may be stored, routed, updated, or controlled by more than one server 120. In particular, the systems and methods described herein may be executed on one or more server 120.

Also shown in FIG. 1B are a plurality of devices 130, shown illustratively and without limitation as device 130-1, 130-1, . . . 130-N. System 100 may thus direct communication between individual server(s) 120 and specific device(s) 130.

As described subsequently, individual devices 130 may be provided with program instructions which may be stored in each device's memory 133 and where the instructions are executed by each device's processor 135. Thus, for example, server(s) 120 may provide device(s) 130 with programming in response to the input of the uses of the individual devices. Further, different devices 130 may be operable using different sets of instructions, that is having one of a variety of different



US 9,063,755 B2

5

“device platforms.” Differing device platforms may result, for example and without limitation, to different operating systems, different versions of an operating system, or different versions of virtual machines on the same operating system. In some embodiments, devices **130** are provided with some programming from authoring system **100** that is particular to the device.

In one embodiment, system **100** provides permits a user of authoring platform **110** to provide instructions to each of the plurality of devices **130** in the form of a device- or device-platform specific instructions for processor **135** of the device, referred to herein and without limitation as a “Player,” and a device-independent program, referred to herein and without limitation as an “Application” Thus, for example, authoring platform **110** may be used to generate programming for a plurality of devices **130** having one of several different device platforms. The programming is parsed into instructions used by different device platforms and instructions that are independent of device platform. Thus in one embodiment, device **130** utilizes a Player and an Application to execute programming from authoring platform **110**. A device having the correct Player is then able to interpret and be programmed according to the Application.

In one alternative embodiment, the Player is executed the first time by device **130** (“activated”) through an Application directory. In another alternative embodiment, the Player is activated by a web browser or other software on device **130**. In yet another alternative embodiment, Player is activated through a signal to device **130** by a special telephone numbers, such as a short code.

When the Application and the Player are provided to memory **133**, the functioning of device **130** may occur in accordance with the desired programming. Thus in one embodiment, the Application and Player includes programming instructions which may be stored in memory **133** and which, when executed by processor **135**, generate the designed displays on screen **137**. The Application and Player may also include programming instructions which may be stored in memory **133** and which provide instructions to processor **135** to accept input from input device **139**.

Authoring tool **112** may, for example, produce and store within memory **111a** plurality of Players (for different devices **130**) and a plurality of Applications for displaying pages on all devices. The Players and Applications are then stored on one or more servers **120** and then provided to individual devices **130**. In general, Applications are provided to device **130** for each page of display or a some number of pages. A Player need be provided once or updated as necessary, and thus may be used to display a large number of Applications. This is advantageous for the authoring process, since all of the device-dependent programming is provided to a device only once (or possibly for some small number of upgrades), permitting a smaller Application, which is the same for each device **130**.

Thus, for example and without limitation, in one embodiment, the Player transforms device-independent instructions of the Application into device-specific instructions that are executable by device **130**. Thus, by way of example and without limitation, the Application may include Java programming for generating a display on screen **137**, and the Player may interpret the Java and instruct processor **135** to produce the display according to the Application for execution on a specific device **130** according to the device platform. The Application may in general include, without limitation, instructions for generating a display on screen **137**, instructions for accepting input from input device **139**, instructions

6

for interacting with a user of device **130**, and/or instructions for otherwise operating the device, such as to place a telephone call.

The Application is preferably code in a device-independent format, referred to herein and without limitation as a Portable Description Language (PDL). The device’s Player interprets or executes the Application to generate one or more “pages” (“Applications Pages”) on a display as defined by the PDL. The Player may include code that is device-specific—that it, each device is provided with a Player that is used in the interpretation and execution of Applications. Authoring tool **112** may thus be used to design one or more device-independent Applications and may also include information on one or more different devices **130** that can be used to generate a Player that specific devices may use to generate displays from the Application.

In one embodiment, system **100** provides Players and Applications to one server **120**, as in FIG. 1A. In another embodiment, system **100** provides Players to a first server **120a** and Applications to a second server **120b**, as in FIG. 1B.

In one embodiment, authoring tool **112** may be used to program a plurality of different devices **130**, and routines **114** may include device-specific routines. In another embodiment, the Player is of the type that is commonly referred to as a “thin client”—that is, software for running on the device as a client in client-server architecture with a device network which depends primarily on a central server for processing activities, and mainly focuses on conveying input and output between the user and the server.

In one embodiment, authoring platform **110** allows user to arrange objects for display on screen. A graphical user interface (“GUI,” or “UI”) is particularly well suited to arranging objects, but is not necessary. The objects may correspond to one or more of an input object, an output object, an action object, or may be a decorative display, such as a logo, or background color or pattern, such as a solid or gradient fill. In another embodiment, authoring platform **110** also permits a user to assign actions to one or more of an input object, an output object, or an action object. In yet another embodiment, authoring platform **110** also permits a user to bind one or more of an input object, an output object, or an action object with web services or web components, or permits a user to provide instructions to processor **135** to store or modify information in memory **133**, to navigate to another display or service, or to perform other actions, such as dialing a telephone number.

In certain embodiments, the applicant model used in developing and providing Applications is a PDL. The PDL can be conceptually viewed as a device, operating system and virtual machine agnostic representation of Java serialized objects. In certain embodiments, the PDL is the common language for authoring tool **112**, the Application, and Player. Thus while either designing the Application with the authoring tool **112**, or programming with the SDK, the internal representation of the programming logic is in Java. In one embodiment the SDK is used within a multi-language software development platform comprising an IDE and a plug-in system to extend it, such as the Eclipse Integrated Development Environment (see, for example, <http://www.eclipse.org/>). At publish time the Java code is translated into a PDL. This translation may also occur in real-time during the execution of any Web Services or backend business logic that interacts with the user.

One embodiment for compacting data that may be used is described in U.S. Pat. No. 6,546,397 to Rempell (“Rempell”), the contents of which are incorporated herein by reference. In that patent the compressed data is described as being a data-

US 9,063,755 B2

7

base. The terminology used here is a PDL, that is the “internal database” of Rempell is equivalent to the PDL of the present Application.

The use of a PDL, as described in Rempell, permits for efficient code and data compaction. Code, as well as vector, integer and Boolean data may be compacted and then compressed resulting in a size reduction of 40 to 80 times that of the original Java serialized objects. This is important not only for performance over the network but for utilizing the virtual memory manager of the Player more efficiently. As an example, the reassembled primitives of the Java objects may first undergo logical compression, followed by LZ encoding.

The use of a PDL also provides virtual machine and operating system independence. Since the reassembled primitives of the Application no longer have any dependencies from the original programming language (Java) that they were defined in. The PDL architecture takes full advantage of this by abstracting all the virtual machine and/or operating system interfaces from the code that processes the PDL.

In one embodiment, the PDL is defined by the means of nested arrays of primitives. Accordingly, the use of a PDL provides extensibility and compatibility, with a minimal amount of constraints in extending the Player seamlessly as market demands and device capabilities continue to grow. Compatibility with other languages is inherent based on the various Player abstraction implementations, which may be, for example and without limitation, Java CDC, J2SE or MIDP2 implementations.

In one embodiment, the architecture of Player P includes an abstraction interface that separates all device, operating system and virtual machine dependencies from the Player's Application model business logic (that is, the logic of the server-side facilities) that extend the Application on the Player so that it is efficiently integrated into a comprehensive client/server Application. The use of an abstraction interface permits the more efficient porting to other operating systems and virtual machines and adding of extensions to the Application model so that a PDL can be implemented once and then seamlessly propagated across all platform implementations. The Application model includes all the currently supported UI objects and their attributes and well as all of the various events that are supported in the default Player. Further, less robust platforms can be augmented by extending higher end capabilities inside that platform's abstraction interface implementation.

In one embodiment, authoring platform 110 provides one or more pages, which may be provided in one Application, or a plurality of Applications, which are stored in memory 123 and subsequently provided to memory 133. In certain embodiments, the Application includes instructions R to request content or web services C from content server 140. Thus, for example and without limitation, the request is for information over the network via a web service, and the request R is responded to with the appropriate information for display on device 130. Thus, for example, a user may request a news report. The Application may include the layout of the display, including a space for the news, which is downloaded from content server 140 for inclusion on the display. Other information that may be provided by content server 140 may include, but is not limited to, pages, Applications, multimedia, and audio.

FIG. 2A is a schematic of a system 200 of an embodiment of system 100 illustrating the communications between different system components. System includes a response director 210, a web component registry 220, and a web service 230. System 200 further includes authoring platform 110, server

8

120, device 130 and content server 140 are which are generally similar to those of the embodiments of FIGS. 1A and 1B, except as explicitly noted.

Response director 210 is a computer or computer system that may be generally similar to server 120 including the ability to communicate with authoring platform 110 and one or more devices 130. In particular, authoring platform 110 generates one or more Players (each usable by certain devices 130) which are provided to response director 210. Devices 130 may be operated to provide response director 210 with a request for a Player and to receive and install the Player. In one embodiment, device 130 provides response director 210 with device-specific information including but not limited to make, model, and/or software version of the device. Response director 210 then determines the appropriate Player for the device, and provides the device with the Player over the network.

Web service 230 is a plurality of services obtainable over the Internet. Each web service is identified and/or defined as an entry in web component registry 230, which is a database, XML file, or PDL that exists on a computer that may be a server previously described or another server 120. Web component registry 230 is provided through server 120 to authoring platform 110 so that a user of the authoring platform may bind web services 230 to elements to be displayed on device 130, as described subsequently.

In one embodiment, authoring platform 110 is used in conjunction with a display that provides a WYSIWYG environment in which a user of the authoring platform can produce an Application and Player that produces the same display and the desired programming on device 130. Thus, for example, authoring tool 112 provides a display on screen 115 that corresponds to the finished page that will be displayed on screen 137 when an Application is intercepted, via a Player, on processor 135 of device 130.

Authoring platform 110 further permits a user of the authoring platform to associate objects, such as objects for presenting on screen 137, with components of one or more web services 230 that are registered in web component registry 220. In one embodiment, information is provided in an XML file to web component registry 220 for each registered components of each web service 230. Web component registry 220 may contain consumer inputs related to each web service 230, environmental data such as PIM, time or location values, persistent variable data, outputs related to the web service, and/or optional hinting for improving the user's productivity.

A user of authoring platform 110 of system 200 may define associations with web services as WebComponent Bindings. In one embodiment, authoring platform 110 allows a user to associate certain objects for display that provide input or output to components of web service 230. The associated bindings are saved as a PDL in server 120.

In one embodiment, an XML web component registry 220 for each registered web service 230 is loaded into authoring platform 110. The user of system 200 can then assign components of any web service 230 to an Application without any need to write code. In one embodiment, a component of web service 230 is selected from authoring platform 110 which presents the user with WYSIWYG dialog boxes that enable the binding of all the inputs and outputs of component of web service 230 to a GUI component of the Application as will be displayed on screen 137. In addition, multiple components of one or more web service 230 can be assigned to any Object or Event in order to facilitate mashups. These Object and/or Event bindings, for each instance of a component of any web service 230, are stored in the PDL. The content server 140

US 9,063,755 B2

9

handles all communication between device **130** and the web service **230** and can be automatically deployed as a web application archive to any content server.

Device **130**, upon detecting an event in which a component of a web service **230** has been defined, assembles and sends all related inputs to content server **240**, which proxies the request to web service **230** and returns the requested information to device **130**. The Player on device **130** then takes the outputs of web service **230** and binds the data to the UI components in the Application, as displayed on screen **137**.

In one embodiment, the mechanism for binding the outputs of the web service to the UI components is through symbolic references that matches each output to the symbolic name of the UI component. The outputs, in one embodiment, may include meta-data which could become part of the inputs for subsequent interactions with the web service.

For example, if a user of authoring platform **110** wants to present an ATOM feed on device **130**, they would search through a list of UI Components available in the authoring platform, select the feed they want to use, and bind the output of the feed summary to a textbox. The bindings would be saved into the PDL on server **120** and processed by device **130** at runtime. If the ATOM feed does not exist a new one can be added to the web component registry that contains all the configuration data required, such as the actual feed URL, the web component manager URL, and what output fields are available for binding.

In another embodiment, components of web services **230** are available either to the user of authoring platform **110** or otherwise accessible through the SDK and Java APIs of routines **114**. System **200** permits an expanding set of components of web services **230** including, but not limited to: server pages from content server **120**; third-party web services including, but not limited to: searching (such through Google or Yahoo), maps (such as through MapQuest and Yahoo), storefronts (such as through ThumbPlay), SMS share (such as through clickatel), stock quotes, social networking (such as through FaceBook), stock quotes, weather (such as through Accuweather) and/or movie trailers. Other components include web services for communication and sharing through chats and forums and rich messaging alerts, where message alerts are set-up that in turn could have components of Web Services **230** defined within them, including the capture of consumer generated and Web Service supplied rich media and textual content.

System **200** also permits dynamic binding of real-time content, where the inputs and outputs of XML web services are bound to GUI components provided on screen **137**. Thus, for example, a user of authoring platform **110** may bind attributes of UI Objects to a particular data base field on a Server. When running the Application, the current value in the referenced data base will be immediately applied. During the Application session, any other real time changes to these values in the referenced data base will again be immediately displayed.

As an example of dynamic binding of real-time content, an RSS feeds and other forms of dynamic content may be inserted into mobile Applications, such as device **130**, using system **200**. Authoring platform **110** may include a "RSS display" list which permits a user to select RSS channels and feeds from an extensible list of available dynamic content. Meta data, such as titles, abstracts and Images can be revealed immediately by the user as they traverse this RSS display list, bringing the PC experience completely and conveniently to mobile devices **130**. In addition, Authoring platform **110** may include a dialog box that dynamically links objects to data and feeds determined by RSS and chat databases. Any relevant attribute for a page view and/or object can be dynamically bound to a value in a server-side database. This includes elements within complex objects such as: any icon or text

10

element within a graphical list; any icon within a launch strip; any feature within any geographical view of a GIS service object; and/or any virtual room within a virtual tour.

As an example of third-party web services **230** that may be provided using system **200**, a user of authoring platform **110** can place, for example, Yahoo maps into device **130** by binding the required component of the Yahoo Maps Web Service, such as Yahoo Map's Inputs and/or Outputs to appropriate Objects of authoring platform **110**. System **200** also provides binding to web services for text, image and video searching by binding to components of those web services.

In one embodiment, an Application for displaying on device **130** includes one or more Applications Pages, each referred to herein as an "XSP," that provides functionality that extends beyond traditional web browsers. The XSP is defined as a PDL, in a similar manner as any Application, although it defines a single page view, and is downloaded to the Player dynamically as required by the PDL definition of the Application. Thus, for example, while JSPs and ASPs, are restricted to the functionality supported by the web browser, the functionality of XSPs can be extended through authoring platform **110** having access to platform dependent routines **114**, such as Java APIs. Combined with dynamic binding functionality, an XSP, a page can be saved as a page object in an author's "pages" library, and then can be dynamically populated with real-time content simultaneously as the page is downloaded to a given handset Player based on a newly expanded API. XSP Server Pages can also be produced programmatically, but in most cases authoring platform **110** will be a much more efficient way to generate and maintain libraries of dynamically changing XSPs.

With XSPs, Applications Pages that have dynamic content associated with them can be sent directly to device **130**, much like how a web browser downloads an HTML page through a external reference. Without XSPs, content authors would have to define each page in the Application. With XSPs, no pages need to be defined. Thus, for example, in a World Cup Application, one page could represent real-time scores that change continuously on demand. With polling (for example, a prompt to the users asking who they predict will win a game), a back-end database would tabulate the information and then send the results dynamically to the handsets. With a bar chart, the Application would use dynamic PDL with scaling on the fly. For example, the server would recalibrate the bar chart for every ten numbers.

Other combinations of components of web services **230** include, but are not limited to, simultaneous video chat sessions, inside an integrated page view, with a video or television station; multiple simultaneous chat sessions, each with a designated individual and/or group, with each of the chat threads visible inside an integrated page view.

Another extension of an XSP is a widget object. Widgets can be developed from numerous sources including, but not limited to, authoring platform **110**, a Consumer Publishing Tool, and an XML to Widget Conversion Tool where the SDK Widget Libraries are automatically populated and managed, or Widget Selection Lists that are available and can be populated with author defined Icons.

Applications, Players, and Processing in a Device

FIG. 2B is a schematic of one embodiment of a device **130** illustrating an embodiment of the programming generated by authoring platform **110**. Memory **133** may include several different logical portions, such as a heap **133a**, a record store **133b** and a filesystem (not shown).

As shown in FIG. 2B, heap **133a** and record store **133b** include programming and/or content. In general, heap **133a** is readily accessible by processor **135** and includes, but is not limited to portions that include the following programming: a portion **133a1** for virtual machine compliant objects representing a single Page View for screen **137**; a portion **133a2** for

US 9,063,755 B2

11

a Player; a portion **133a3** for a virtual machine; and a portion **133a4** for an operating system.

Record store **133b** (or alternatively the filesystem) includes, but is not limited to, portions **133b1** for Applications and non-streaming content, which may include portions **133a2** for images, portions **133a4** for audio, and/or portions **133a5** for video, and portions **133b2** for non-Application PDLs, such as a Master Page PDL for presenting repeating objects, and Alerts, which are overlayed on the current page view. Other content, such as streaming content may be provided from network interface **131** directly to the Media Codec of device **130** with instructions from Player on how to present the audio or video.

In one embodiment, the Player includes a Threading Model and a Virtual Memory Manager. The Threading Model first manages a queue of actions that can be populated based on Input/Output events, Server-side events, time-based events, or events initiated by user interactions. The Threading Model further manages the simultaneous execution of actions occurring at the same time. The Virtual Memory Manager includes a Logical Virtual Page controller that provides instructions from the record store to the heap, one page at a time. Specifically, the Virtual Memory Manager controls the transfer of one of the Application Pages and its virtual machine compliant objects into portion **133a1** as instructions readable by the Player or Virtual Machine. When the Player determines that a new set of instructions is required, the information (such as one Application Page is retrieve from the Record store, converted into virtual machine compliant objects (by processor **135** and according to operation by the Player, Virtual Machine, etc). and stored in heap **133a**. Alternatively, the Player may augment virtual machine compliant objects with its own libraries for managing user interactions, events, memory, etc.

The connection of portions **133a1**, **133a2**, **133a3**, **133a4**, record store **133b** and processor **135** are illustrative of the logical connection between the different types of programming stored in Heap **133a** and record store **133b**, that is, how data is processed by processor **135**.

The Player determines which of the plurality of Application Pages in portion **133b1** is required next. This may be determined by input actions from the Input Device **139**, or from instructions from the current Application Page. The Player instructs processor **135** to extract the PDF from that Applications Page and store it in portion **133a1**. The Player then interprets the Application Page extracted from PDL which in turn defines all of the virtual machine compliant Objects, some of which could have attributes that refer to images, audio, and/or video stored in portions **133a3**, **133a4**, **133a5**, respectively.

The Virtual Machine in portion **133a3** processes the Player output, the Operating System in portion **133a3** processes the Virtual Machine output which results in machine code that is processed by the Operating System in portion **133a4**.

In another embodiment, the Player is a native program that interacts directly with the operating system.

Embodiments of a Publishing Environment

In one embodiment, authoring platform **110** includes a full-featured authoring tool **112** that provides a what-you-see-is-what-you-get (WYSIWYG) full featured editor. Thus, for example, authoring tool **112** permits a user to design an Application by placing objects on canvas **305** and optionally assigning actions to the objects and save the Application. System **100** then provides the Application and Player to a device **130**. The Application as it runs on device **130** has the same look and operation as designed on authoring platform

12

**110**. In certain embodiments, authoring platform **110** is, for example and without limitation, a PC-compatible or a Macintosh computer.

Authoring platform **110** produces an Application having one or more Applications Pages, which are similar to web pages. That is, each Applications Page, when executed on device **130** may, according to its contents, modify what is displayed on screen **137** or cause programming on the device to change in a manner similar to how web pages are displayed and navigated through on a website.

In one embodiment, authoring tool **112** allows a user to place one or more objects on canvas **305** and associate the objects with an Applications Pages. Authoring platform **110** maintains a database of object data in memory **111**, including but not limited to type of object, location on which page, and object attributes. The user may add settings, events, animations or binding to the object, from authoring tool **112**, which are also maintained in memory **111**. Authoring tool **112** also allows a user to define more than one Applications Page.

In another embodiment, authoring tool **112**, provides Java programming functions of the Java API for specific devices **130** as pull-down menus, dialog boxes, or buttons. This permits a user of authoring platform **110** to position objects that, after being provided as an Application to device **130**, activate such Java functions on the device.

In certain embodiments, authoring platform **110**, as part of system **100**, permits designers to include features of advanced web and web services Applications for access by users of device **130**. Some of the features of advanced web and web services include, but are not limited to: slide shows, images, video, audio, animated transitions, multiple chats, and mouse interaction; full 2-D vector graphics; GIS (advanced LBS), including multiple raster and vector layers, feature sensitive interactions, location awareness, streaming and embedded audio/video, virtual tours, image processing and enhancement, and widgets. In other embodiments the features are provided for selection in authoring platform **110** through interactive object libraries.

In certain embodiments, authoring platform **110**, as part of system **100**, allows the inclusion of child objects which may eventually be activated on device **130** by the user of the device or by time. The uses of the child objects on device **130** include, but are not limited to: mouse over (object selection), hover and fire events and launching of object-specific, rich-media experiences.

In certain other embodiments, authoring platform **110**, as part of system **100**, provides advanced interactive event models on device **130**, including but not limited to: user-, time- and/or location-initiated events, which allow content developers to base interactivity on specific user interactions and/or instances in time and space; timelines, which are critical for timing of multiple events and for animations when entering, on, or exiting pages of the Application; waypoints, which act similar to key frames, to allow smooth movement of objects within pages of the Application. Waypoints define positions on a page object's animation trajectory. When an object reaches a specific waypoint other object timelines can be initiated, creating location-sensitive multiple object interaction, and/or audio can be defined to play until the object reaches the next waypoint.

Authoring platform **110** may also define a Master Page, which acts as a template for an Applications Page, and may also define Alert Pages, which provide user alerts to a user of device **130**.

In certain embodiments, authoring platform **110**, as part of system **100**, provides full style inheritance on device **130**. Thus, for example and without limitation, both master page

US 9,063,755 B2

13

inheritance (for structural layout inheritance and repeating objects) and object styles (for both look and feel attribute inheritance) are supported. After a style has been defined for an object, the object will inherit the style. Style attributes include both the look and the feel of an object, including mouse interaction, animations, and timelines. Each page may include objects that may be a parent object or a child object. A child object is one that was created by first selecting a parent object, and then creating a child object. Child objects are always part of the same drawing layer as its parent object, but are drawn first, and are not directly selectable when running the Application. A parent object is any object that is not a child object, and can be selected when running the Application.

As an example, the user of authoring tool 112 may create various text objects on canvas 305 using a style that sets the font to red, the fonts of these objects will be red. Suppose user of authoring tool 112 changes the font color of a specific button to green. If later, the user of authoring tool 112 changes the style to blue; all other text objects that were created with that style will become blue except for the button that had been specifically set to green.

In certain other embodiments, authoring platform 110 provides page view, style, object, widget and Application template libraries. Authoring platform 110 may provide templates in private libraries (available to certain users of the authoring platform) and public libraries (available to all users of the authoring platform). Templates may be used to within authoring platform 110 to define the look and feel of the entire Application, specific pages, or specific slide shows and virtual tours as seen on device 130.

FIGS. 3A and 3B illustrate one embodiment of a publisher interface 300 as it appears, for example and without limitation, on screen 115 while executing authoring tool 112. In one embodiment, publisher interface 300 includes a Menu bar 301, a Tool bar 303, a Canvas 305, a Layer Inspector 307 having subcomponents of a page/object panel 307a, an object style panel 307b, and a page alert panel 307c, and a Resource Inspector 309.

In general, publisher interface 300 permits a user of authoring platform 110 to place objects on canvas 305 and then associate properties and/or actions to the object, which are stored in the Application. As described subsequently, publisher interface 300 permits a user to program a graphical interface for the screen 137 of device 130 on screen 115 of authoring platform 110, save an Application having the programming instructions, and save a Player for the device. The intended programming is carried out on device 130 when the device, having the appropriate device platform Player, receives and executes the device-independent Application.

Thus, for example, authoring tool 112 maintains, in memory 111, a list of every type of object and any properties, actions, events, or bindings that may be assigned to that object. As objects are selected for an Application, authoring tool 112 further maintains, in memory 111, a listing of the objects. As the user selects objects, publisher interface 300

14

provides the user with a choice of further defining properties, actions, events, or bindings that may be assigned to each particular object, and continues to store the information in memory 111.

In one embodiment, publisher interface 300 is a graphical interface that permits the placement and association of objects in a manner typical of, for example, vector graphics editing programs (such as Adobe Illustrator). Objects located on canvas 305 placed and manipulated by the various commands within publisher interface 300 or inputs such as an input device 117 which may be a keyboard or mouse. As described herein, the contents of canvas 305 may be saved as an Application that, through system 100, provide the same or a similar placement of objects on screen 137 and have actions defined within publisher interface 300. Objects placed on canvas 305 are intended for interaction with user of device 130 and are referred to herein, without limitation, as objects or UI (user interface) objects. In addition, the user of interface 300 may assign or associate actions or web bindings to UI objects placed on canvas 305 with result in the programming device 130 that cause it to respond accordingly.

Objects include, but are not limited to input UI objects, response UI objects. Input UI objects include but are not limited to: text fields (including but not limited to alpha, numeric, phone number, or SMS number); text areas; choice objects (including but not limited to returning the selected visible string or returning a numeric hidden attribute); single item selection lists (including but not limited to returning the selected visible string or returning a numeric hidden attribute); multi item selection lists (including but not limited to returning all selected items (visible text string or hidden attribute) or cluster item selection lists (returning the hidden attributes for all items).

Other input UI objects include but are not limited to: check boxes; slide show (including but not limited to returning a numeric hidden attribute, returning a string hidden attribute, or returning the hidden attributes for all slides); and submit function (which can be assigned to any object including submit buttons, vectors, etc.).

Response UI Objects may include, but are not limited to: single line text objects, which include: a text Field (including but not limited to a URL, audio URL, or purchase URL), a text button, a submit button, or a clear button. Another response UI objects include: a multiple line text object, which may include a text area or a paragraph; a check box; an image; a video; a slide show (with either video or image slides, or both); choice objects; list objects; or control lists, which control all the subordinate output UI objects for that web component. Control list objects include, but are not limited to: list type or a choice type, each of which may include a search response list or RSS display list.

As a further example of objects that may be used with authoring tool 112, Table I lists Data Types, Preferred Input, Input Candidates, Preferred Output and Output Candidates for one embodiment of an authoring tool.

TABLE I

One embodiment of supported objects				
Data Types	Preferred Input	Input Candidates	Preferred Output	Output Candidates
boolean	Check Box	Check Box	Check Box	Check Box
Int	Text Field (integer)	Text Field (integer)	Text Field (integer)	Text Field (integer)
		Text Field (Phone #)		Text Field (Phone #)
		Text Field (SMS #)		Text Field (SMS #)
		Choice		Choice
		List (single select)		List (single select)
				Text Button

US 9,063,755 B2

15

16

TABLE I-continued

One embodiment of supported objects				
Data Types	Preferred Input	Input Candidates	Preferred Output	Output Candidates
String	Text Field (Alpha)	Any	Text Field (Alpha)	Any
multilineString	Text Area	Text Area	Text Area	Text Area
ImageURL	N/A	N/A	Image	Image
VideoURL	N/A	N/A	Video	Slide Show
List	Single Item List	Single Item List	Single Item List	Video
		Multi-Select List		Slide Show
		Complex List		Any List Type
		Choice		Any Choice Type
		Slide Show		(see Complex List Specification)
ComplexList	Complex List	Single Item List	Single Item List	Any List Type
		Multi-Select List		(see Complex List Specification)
		Complex List		Slide Show
Slideshow	Slide Show	Slide Show	Slide Show	Search Response List
SearchResponseList	N/A	N/A	Search Response List	Control List
				Complex List
				Choice
RSSList	N/A	N/A	RSS Display List	RSS Display List
				Control List
				Complex List
				Choice
SingleSelectionList	Choice	Choice	Choice	Choice
		Complex List		Complex List
MultiSelectionList	Multi-Selection List	Multi-Selection List	Multi-Selection List	Multi-Selection List
ServiceActivation	Submit Button	Any	N/A	N/A
ChannelImageURL	N/A	N/A	Image	Image
				Video
				Slide Show
ChannelDescription	N/A	N/A	Text Area	Text Area
				Paragraph
				Text Field
				Text Button
				List
				Choice
ChannelTitle	N/A	N/A	Text Field	Text Field
				Text Button
				Paragraph
				Text Area
				List
				Choice
URL			Text Field	Text Field
			(URL request)	(URL request)
Audio URL			Text Field	Text Field
			(Audio URL request)	(Audio URL request)
Purchase URL			Text Field	Text Field
			(Purchase URL request)	(Purchase URL request)
Image Data			Image	Image
				Slide Show
Image List Data			Slide Show	Slide Show
				Image
Persistent Variable	N/A	N/A	N/A	N/A
Pipeline Multiple Select	Multi-select List	Multi-select List	N/A	N/A
		Complex List		
		Slide Show		
Phone Number	Text Field	Text Field	Text Field	Text Field
	(numeric type)	Text Button	(numeric type)	Text Button
Hidden Attribute	Complex List	Complex List	Complex List	Complex List
		Slide Show		Slide Show
Collection List	N/A	N/A	Slide Show	Complex List
				Slide Show

In general, publisher interface **300** permits a user to define an Application as one or more Applications Pages, select UI objects from Menu bar **301** or Tool bar **303** and arrange them on an Applications Page by placing the objects on canvas **305**. An Application Page is a page that is available to be visited through any navigation event. Application Pages inherit all the attributes of the Master Page, unless that attribute is specifically changed during an editing session.

Authoring platform **110** also stores information for each UI object on each Application Page of an Application. Layer Inspector **307** provides lists of Applications Pages, UI objects on each Applications Page, and Styles, including templates. Objects may be selected from canvas **305** or Layer Inspector **307** causing Resource Inspector **309** to provide lists of various UI objects attributes which may be selected from within the Resource Inspector. Publisher interface **300** also permits a user to save their work as an Application for layer transfer and

US 9,063,755 B2

17

operation of device **130**. Publisher interface **300** thus provides an integrated platform for designing the look and operation of device **130**.

The information stored for each UI object depends, in part, on actions which occur as the result of a user of device **130** selecting the UI object from the device. UI objects include, but are not limited to: navigational objects, such as widget or channel launch strips or selection lists; message objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields or a pop-up alert; text fields or areas; check boxes; pull down menus; selection lists and buttons; pictures; slide shows; video or LBS maps; shapes or text defined by a variety of tools; a search response; or an RSS display.

In certain embodiments, publisher interface **300** permits a user to assign action to UI objects, including but not limited to, programming of the device **130** or a request for information over network N. In one embodiment, for example and without limitation, publisher interface **300** has a selection to bind a UI object to a web service—that is, associate the UI object or a manipulation or selection of UI object with web services. Publisher interface **300** may also include many drawing and text input functions for generating displays that may be, in some ways, similar to drawing and/or word processing programs, as well as toolbars and for zooming and scrolling of a workspace.

Each UI object has some form, color, and display location associate with it. Further, for example and without limitation, UI objects may have navigational actions (such as return to home page), communications actions (such as to call the number in a phone number field), or web services (such as to provide and/or retrieve certain information from a web service). Each of these actions requires authoring platform **110** to store the appropriate information for each action. In addition, UI objects may have associated parent or child objects, default settings, attributes (such as being a password or a phone number), whether a field is editable, animation of the object, all of which may be stored by authoring platform **110**, as appropriate.

Menu bar **301** provides access features of publisher interface **300** through a series of pull-down menus that may include, but are not limited to, the following pull-down menus: a File menu **301a**, an Edit menu **301b**, a View menu **301c**, a Project menu **301d**, an Objects menu **301e**, an Events menu **301f**, a Pages menu **301g**, a Styles menu **301h**, and a Help menu **301i**.

File menu **301a** provides access to files on authoring platform **110** and may include, for example and without limitation, selections to open a new Application or master page, open a saved Application, Application template, or style template, import a page, alert, or widget, open library objects including but not limited to an image, video, slide show, vector or list, and copying an Application to a user or to Server **120**.

Edit menu **301b** may include, but is not limited to, selections for select, cut, copy, paste, and edit functions.

View menu **301c** may include, but is not limited to, selections for zooming in and out, previewing, canvas **305** grid display, and various palette display selections.

Project menu **301d** may include, but is not limited to, selections related to the Application and Player, such as selections that require a log in, generate a universal Player, generate server pages, activate server APIs and extend Player APIs. A Universal Player will include all the code libraries for the Player, including those that are not referenced by the current

18

Application. Server APIs and Player APIs logically extend the Player with Server-side or device-side Application specific logic.

Objects menu **301e** includes selections for placing various objects on canvas **305** including, but not limited to: navigation UI objects, including but not limited to widget or channel launch strips or selection lists; message-related UI objects, including but not limited to multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert; shapes, which provides for drawing tools; forms-related objects, including but not limited to text fields; scrolling text box, check box, drop-down menu, list menu, submit button or clear button; media-related UI objects such as pictures, slide shows, video or LBS maps; text-related UI objects such as buttons or paragraphs; and variables, including but not limited to time, date and audio mute control.

Events menu **301f** includes selections for defining child objects, mouse events, animations or timelines.

Pages menu **301g** includes selection for handling multi-page Applications, and may include selections to set a master page, delete, copy, add or go to Applications Pages.

Styles menu **301h** includes selections to handle styles, which are the underlying set of default appearance attributes or behaviors that define any object that is attached to a style. Styles are a convenient way for quickly creating complex objects, and for changing a whole collection of objects by just modifying their common style. Selections of Styles menu **301h** include, but not limited to, define, import, or modify a style, or apply a template. Help menu **301i** includes access a variety of help topics.

Tool bar **303** provides more direct access to some of the features of publisher interface **300** through a series of pull-down menus. Selections under tool bar **303** may include selections to:

control the look of publisher interface **300**, such as a Panel selection to control the for hiding or viewing various panels on publisher interface **300**;

control the layout being designed, such as an Insert Page selection to permit a user to insert and name pages;

control the functionality of publisher interface **300**, such as a Palettes selection to choose from a variety of specialized palettes, such as a View Palette for zooming and controlling the display of canvas **305**, a Command Palette of common commands, and Color and Shape Palettes;

place objects on canvas **305**, which may include selections such as: a Navigation selection to place navigational objects, such as widget or channel launch strips or selection lists), a Messages selection to place objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert, a Forms selection to place objects such as text fields or areas, check boxes, pull down menus, selection lists, and buttons, a Media selection to place pictures, slide shows, video or LBS maps, and a Shapes selection having a variety of drawing tools, a Text selection for placing text, a search response, or an RSS display, and Palettes.

In one embodiment, Tool bar **303** includes a series of pull-down menus that may include, but are not limited to, items from Menu bar **301** organized in the following pull-down menus: a Panel menu **303a**, an Insert Page menu **303b**, a Navigation menu **303c**, a Messages menu **303d**, a Forms menu **303e**, a Media menu **303f**, a Shapes menu **303g**, a Text menu **303h**, and a Palettes menu **303i**.

Panel menu **303a** permits a user of authoring platform **110** to change the appearance of interface **300** by, controlling which tools are on the interface or the size of canvas **305**. Insert Page menu **303b** permits a user of authoring platform **110** to open a new Application Page. Navigation menu **303c**

US 9,063,755 B2

19

displays a drop down menu of navigational-related objects such as a widget or channel launch strip or selection list. Messages menu **303d** displays a drop down menu of messaging-related objects such as multiple chat, video chat, phone or SMS lists or fields, and pop-up alerts. Forms menu **303e** displays a drop down menu of forms-related objects including, but not limited to, a text field, a text area, a check box, a drop down menu, a selection list, a submit button, and a clear button. Media menu **303f** displays a drop down menu of media-related objects including, but not limited to, a picture, slide show, video or LBS map. Shapes menu **303g** displays a drop down menu of draw tools, basic shapes, different types of lines and arrows and access to a shape library. Text menu **303j** displays a drop down menu of text-related objects, including but not limited to a text button, paragraph, search response, RSS display and variables such as time and date.

Palettes menu **301i** includes a selection of different palettes that can be moved about publisher interface **300**, where each palette has specialized commands for making adjustments or associations to objects easier. Palettes include, but are not limited to: a page view palette, to permit easy movement between Applications Pages; a view palette, to execute an Application or zoom or otherwise control the viewing of an Application; a commands palette having editing commands; a color palette for selection of object colors; and a shapes palette to facilitate drawing objects.

Layer inspector **307** permits a user of publisher interface **300** to navigate, select and manipulate UI objects on Applications Pages. Thus, for example, a Page/objects panel **307a** of layer inspector **307** has a listing that may be selected to choose an Applications Pages within an Application, and UI objects and styles within an Applications Page. An Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits selection of UI objects for operations to be performed on the objects.

Thus, for example, when objects from Menu bar **301** or Tool bar **303** are placed on canvas **305**, the name of the object appears in Page/objects panel **307a**. Page/objects panel **307a** includes a page display **307a1** and an objects display **307a2**. Page display **307a1** includes a pull down menu listing all Applications Pages of the Application, and objects display **307a2** includes a list of all objects in the Applications Page (that is, objects on canvas **305**).

In general, page/objects panel **307a** displays various associations with a UI object and permits various manipulations including, but not limited to, operations for parent and child objects that are assigned to a page, and operations for object styles, and permits navigating between page types and object styles, such as switching between the master page and Application pages and deselecting object styles and alerts, opening an Edit Styles Dialog Box and deselecting any master, Application or alert page, or selecting an alert page and deselecting any Master Page or Application Page. A parent or child object can also be selected directly from the Canvas. In either case, the Resource Inspector can then be used for modifying any attribute of the selected object.

Examples of operations provided by page/objects panel **307a** on pages include, but are not limited to: importing from either a user's private page library or a public page library; deleting a page; inserting a new page, inheriting all the attributes of the Master Page, and placing the new page at any location in the Page List; editing the currently selected page, by working with an Edit Page Dialog Box. While editing all the functions of the Resource Inspector **309** are available, as described subsequently, but are not applied to the actual page until completing the editing process.

20

Examples of operations provided by of page/objects panel **307a** on objects, which may be user interface (UI) objects, include but are not limited to: changing the drawing order layer to: bring to the front, send to the back, bring to the front one layer, or send to the back one layer; hiding (and then reshowing) selected objects to show UI objects obstructed by other UI Objects, delete a selected UI Page Object, and editing the currently selected page, by working with a Edit Page Dialog Box.

Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits operations to be performed on objects, and is similar to panel **307a**. Examples of operations provided by object style panel **307b** include, but are not limited to: importing from either a user's private object library or a public object library; inserting a new object style, which can be inherited from a currently selected object, or from a previously defined style object; and editing a currently selected object style by working with an Edit Style Dialog Box.

Style attributes can be assigned many attributes, including the look, and behavior of any object that inherits these objects. In addition, List Layout Styles can be created or changed as required. A layout style can define a unbounded set of Complex List Layouts, including but not limited to: the number of lines per item in a list, the number of text and image elements and their location for each line for each item in the last, the color and font for each text element, and the vertical and horizontal offset for each image and text element.

Alerts Panel **307c** provides a way of providing alert pages, which can have many of the attributes of Application Pages, but they are only activated through an Event such as a user interaction, a network event, a timer event, or a system variable setting, and will be superimposed onto whatever is currently being displayed. Alert Pages all have transparent backgrounds, and they function as a template overlay, and can also have dynamic binding to real time content.

Resource inspector **309** is the primary panel for interactively working with UI objects that have been placed on the Canvas **305**. When a UI object is selected on Canvas **305**, a user of authoring platform **110** may associate properties of the selected object by entering or selecting from resource inspector **309**. In one embodiment, resource inspector **309** includes five tab selections: Setting Tab **309a**, Events Tab **309b**, Animation Tab **309c**, Color Tab **309d** which includes a color palette for selecting object colors, and Bindings Tab **309e**.

Settings Tab **309a** provides a dialog box for the basic configuration of the selected object including, but not limited to, name, size, location, navigation and visual settings. Depending upon the type of object, numerous other attributes could be settable. As an example, the Setting Tab for a Text Field may include dialog boxes to define the text field string, define the object style, set the font name, size and effects, set an object name, frame style, frame width, text attributes (text field, password field, numeric field, phone number, SMS number, URL request).

As an example of Setting Tab **309a**, FIG. 3B shows various selections including, but not limited to, setting **309a1** for the web page name, setting **309a2** for the page size, including selections for specific devices **130**, setting **309a3** indicating the width and height of the object, and setting **309a4** to select whether background audio is present and to select an audio file.

FIG. 3C illustrates an embodiment of the Events Tab **309b**, which includes all end user interactions and time based operations. The embodiment of Events Tab **309b** in FIG. 3C includes, for example and without limitation, an Events and



US 9,063,755 B2

21

Services **309b1**, Advanced Interactive Settings **309b2**, Mouse State **309b3**, Object Selected Audio Setting **309b4**, and Work with Child Objects and Mouse Overs button **309b5**.

Events and Services **309b1** lists events and services that may be applied to the selected objects. These include, but are not limited to, going to external web pages or other Applications pages, either as a new page or by launching a new window, executing an Application or JavaScript method, pausing or exiting, placing a phone call or SMS message, with or without single or multiple Player download, show launch strip, or go back to previous page. Examples of events and services include, but are not limited to those listed in Table II

TABLE II

Events and Services	
Goto External Web Page replacing Current Frame	ChoiceObject: Remove Icon from Launch Strip
Goto External Web Page Launched in a New Window	Goto a specific Internal Web Page with Alert.
Goto a specific Internal Web Page	"Backend Synchronization"
Goto the next Internal Web Page	Goto Widget Object
	Generate Alert.
	"With a Fire Event"
Goto External Web Page replacing the Top Frame	Send SMS Message from Linked Text Field
Execute JavaScript Method	Toggle Alert. "Display OnFocus, Hide OffFocus"
Pause/Resume Page Timeout	Execute an Application with Alert. "With a Fire Event"
Execute an Application	Goto Logical First Page
Goto a specific Internal Web Page with setting starting slide	Generate Alert with Backend Synchronization
Exit Application	Send SMS Message with Share (Player Download)
Exit Player	Place PhoneCall from linked Text Field with Share (Player Download)
Place PhoneCall from linked Text Field	Send IM Alert from linked Text Field or Text Area
Text Field/Area: Send String on FIRE	Set and Goto Starting Page
ChoiceObject: Add Icon to Launch Strip	Populate Image
Text Field/Area: Send String on FIRE or Numeric Keys	Preferred Launch Strip

Advanced Interactive Settings **309b2** include Scroll Activation Enabled, Timeline Entry Suppressed, Enable Server Listener, Submit Form, Toggle Children on FIRE, and Hide Non-related Children. Mouse State **309b3** selections are Selected or Fire. When Mouse State Selected is chosen, Object Selected Audio Setting **309b4** of Inactive, Play Once, Loop, and other responses are presented. When Mouse State Fire is chosen, Object Selected Audio Setting **309b4** is replaced with FIRE Audi Setting, with appropriate choices presented.

When Work with Child Objects and Mouse Overs button **309b5** is selected, a Child Object Mode box pops up, allowing a user to create a child object with shortcut to Menu bar **301** actions that may be used define child objects.

FIG. 3D illustrates one embodiment of an Animation Tab **309c**, which includes all animations and timelines. The Color Tab includes all the possible color attributes, which may vary significantly by object type.

Animation Tab **309c** includes settings involved in animation and timelines that may be associated with objects. One embodiment of Animation Tab **309c** is shown, without limitation, in FIG. 3D, and is described, in Rempell ("Rempell").

A Color Tab **309d** includes a color palette for selecting object colors.

Bindings Tab **309e** is where web component operations are defined and dynamic binding settings are assigned. Thus, for

22

example, a UI object is selected from canvas **305**, and a web component may be selected and configured from the bindings tab. When the user's work is saved, binding information is associated with the UI object that will appear on screen **137**.

FIG. 3E illustrates one embodiment of Bindings Tab and includes, without limitation, the following portions: Web Component and Web Services Operations **309e1**, Attributes Exposed list **309e2**, panel **309e3** which includes dynamic binding of server-side data base values to attributes for the selected object, Default Attribute Value **309e4**, Database Name **309e5**, Table Name **309e6**, Field Name **309e7**, Channel Name **309e8**, Channel Feed **309e9**, Operation **309e10**, Select Link **309e11**, and Link Set checkbox **309e12**.

Web Component and Web Services Operations **309e1** includes web components that may be added, edited or removed from a selected object. Since multiple web components can be added to the same object, any combination of mash-ups of 3rd party web services is possible. When the "Add" button of Web Component and Web Services Operations **309e1** is selected, a pop-up menu **319**, as shown in FIG. 3F, appears on publisher interface **300**. Pop-up menu **319** includes, but is not limited to, the options of: Select a Web Component **319a**; Select Results Page **319b**; Activation Options **319c**; Generate UI Objects **319d**; and Share Web Component **319e**.

The Select a Web Component **319a** portion presents a list of web components. As discussed herein, the web components are registered and are obtained from web component registry **220**.

Select Results Page **319b** is used to have the input and output on different pages—that is, when the Results page is different from Input page. The default selected results page is either the current page, or, if there are both inputs and outputs, it will be set provisionally to the next page in the current page order, if one exists.

Activation Options **319c** include, if there are no Input UI Objects, a choice to either "Preload" the web component, similar to how dynamic binding, or have the web component executed when the "Results" page is viewed by the consumer.

Generate UI Objects **319c**, if selected, will automatically generate the UI objects. If not selected, then the author will bind the Web Component Inputs and Results to previously created UI Objects.

Share Web Component **319e** is available and will become selected under the following conditions: 1) Web Component is Selected which already has been used by the current Application; or 2) the current Input page is also a "Result" page for that Web component. This permits the user of device **130**, after viewing the results, to extend the Web Component allowing the user to make additional queries against the same Web Component. Examples of this include, but are not limited to, interactive panning and zooming for a Mapping Application, or additional and or refined searches for a Search Application.

Dynamic Binding permits the binding of real time data, that could either reside in a 3<sup>rd</sup> party server-side data base, or in the database maintained by Feed Collector **1010** for aggregating live RSS feeds, as described subsequently with reference to FIG. 10.

Referring again to FIG. 3E, Attributes Exposed list **309e2** are the attributes available for the selected object that can be defined in real time through dynamic binding.

Panel **309e3** exposes all the fields and tables associated with registered server-side data bases. In one embodiment, the user would select an attribute from the "Attributes Exposed List" and then select a data base, table and field to define the real time binding process. The final step is to define

US 9,063,755 B2

23

the record. If the Feed Collector data base is selected, for example, then the RSS “Channel Name” and the “Channel Feed” drop down menus will be available for symbolically selected the record. For other data bases the RSS “Channel Name” and the “Channel Feed” drop down menus are replaced by a “Record ID” text field.

Default Attribute Value **309e4** indicates the currently defined value for the selected attribute. It will be overridden in real time based on the dynamic linkage setting.

Database Name **309e5** indicates which server side data base is currently selected. Table Name **309e6** indicates which table of the server side data base is currently selected.

Field Name **309e7**, indicates which field form the selected table of the server side data base is currently selected.

Channel Name **309e8** indicates a list of all the RSS feeds currently supported by the Feed Collector. This may be replaced by “Record ID” if a data base other than the Feed Collector **1010** is selected.

Channel Feed **309e9** indicates the particular RSS feed for the selected RSS Channel. Feed Collector **1010** may maintain multiple feeds for each RSS channel.

Operation **309e10**, as a default operation, replaces the default attribute value with the real time value. In other embodiments this operation could be append, add, subtract, multiply or divide.

Select Link **309e11** a button that, when pressed, creates the dynamic binding. Touching the “Select Link” will cause the current data base selections to begin the blink in some manner, and the “Select Link” will change to “Create Link”. The user could still change the data base and attribute choices. Touching the “Create Link” will set the “Link Set” checkbox and the “Create Link” will be replaced by “Delete Link” if the user wishes to subsequently remove the link. When the application is saved, the current active links are used to create the SPDL.

Link Set checkbox **309e12** indicates that a link is currently active.

An example of the design of a display is shown in FIGS. 4A and 4B according to the system **100**, where FIG. 4A shows publisher interface **300** having a layout **410** on canvas **305**, and FIG. 4B shows a device **130** having the resulting layout **420** on screen **137**. Thus, for example, authoring platform **110** is used to design layout **410**. Authoring platform **110** then generates an Application and a Player specific to device **130** of FIG. 4B. The Application and Player are thus used by device **130** to produce layout **420** on screen **137**.

As illustrated in FIG. 4A, a user has placed the following on canvas **305** to generate layout **410**: text and background designs **411**, a first text input box **413**, a second text input box **415**, and a button **417**. As an example which is not meant to limit the scope of the present invention, layout **410** is screen prompts a user to enter a user name in box **413** and a password in box **415**, and enter the information by clicking on button **417**.

In one embodiment, all UI objects are initially rendered as Java objects on canvas **305**. When the Application is saved, the UI objects are transformed into the PDL, as described subsequently.

Thus, for example, layout **410** may be produced by the user of authoring platform **110** selecting and placing a first Text Field as box **413** then using the Resource Inspector **309** portion of interface **300** to define its attributes.

Device User Experience

Systems **100** and **200** provide the ability for a very large number of different types of user experiences. Some of these are a direct result of the ability of authoring platform **110** to bind UI objects to components of web services. The following

24

description is illustrative of some of the many types of experiences of using a device **130** as part of system **100** or **200**.

Device **130** may have a one or more of a very powerful and broad set of extensible navigation objects, as well as object- and pointer-navigation options to make it easy to provide a small mobile device screen **137** with content and to navigate easily among page views, between Applications, or within objects in a single page view of an Application.

Navigation objects include various types of launch strips, various intelligent and user-friendly text fields and scrolling text boxes, powerful graphical complex lists, as well as Desktop-level business forms. In fact, every type of object can be used for navigation by assigning a navigation event to it. The authoring tool offers a list of navigation object templates, which then can be modified in numerous ways.

Launch Strips and Graphical List Templates Launch Strips

Launch strips may be designed by the user of authoring platform **110** with almost no restrictions. They can be stationary or appear on command from any edge of the device, their size, style, audio feedback, and animations can be freely defined to create highly compelling experiences.

FIG. 5 shows a display **500** of launch strips which may be on displayed canvas **305** or on screen **137** of device **130** having the proper Player and Application. Display **501** includes a portal-type Launch Strip **501** and a channel-type Launch Strip **502**, either one of which may be included for navigating the Application.

Launch Strip **501** includes UI objects **501a**, **501b**, **501c**, **501d**, and **501e** that that becomes visible from the left edge of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector **309** with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. When the Applications Page, having been saved by authoring platform **110** and transferred to display **130**, is executed on device **130**, a user of the device may easily navigate the Application.

Launch Strip **502** includes UI objects **502b**, **502c**, **502d**, and **503e** that that becomes visible from the bottom of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector **309** with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. Launch Strip **502** also includes UI objects **502a** and **503g**, which include the graphic of arrows, and which provide access to additional navigation objects (not shown) when selected by a user of device **130**. Launch strip **502** may also include sound effects for each channel when being selected, as well as popup bubble help.

Additional navigational features are illustrated in FIG. 6A as a display of a Channel Selection List **601a**, in FIG. 6B as a display of a Widget Selection List **601b**, and in FIG. 6C as display of a Phone List **601c**. Lists **601a**, **601b**, and **601c** may be displayed on canvas **305** or on screen **137** of device **130** having the proper Player and Application. As illustrated, graphical lists **601a**, **601b**, and **601c** may contain items with many possible text and image elements. Each element can be defined at authoring time and/or populated dynamically through one or more Web Service **250** or API. Assignable Navigation Events. All objects, and/or all elements within an object, can be assigned navigation events that can be extended to registered web services or APIs. For example, a Rolodex-type of navigation event can dynamically set the starting slide of the targeted page view (or the starting view of a targeted Application).

In the embodiment of FIGS. 6A, 6B, and 6C, each list **601a**, **601b**, and **601c** has several individual entries that are

## US 9,063,755 B2

25

each linked to specific actions. Thus Channel Selection List **601a** shows three objects, each dynamically linked to a web service (ESPN, SF 49ers, and Netflix) each providing a link to purchase or obtain items from the Internet. Widget Selection List **601b** includes several objects presenting different widgets for selecting. Phone List **601c** includes a list phone number objects of names that, when selected by a user of device **130** cause the number to be dialed. Entries in Phone List **601c** may be generated automatically from either the user's contact list that is resident on the device, or through a dynamic link to any of user's chosen server-side facilities such as Microsoft Outlook, Google Mail, etc. In one embodiment, Phone List **601c** may be generated automatically using a web component assigned to the Application, which would automatically perform those functions.

In another embodiment, authoring platform **110** allows a navigation selection of objects with a Joy Stick and/or Cursor Keys in all 4 directions. When within a complex object the navigation system automatically adopts to the navigation needs for that object. For coordinate sensitive objects such as geographical information services (GIS) and location-based services (LBS) or virtual tours a soft cursor appears. For Lists, scrolling text areas and chats, Launch strips, and slide shows the navigation process permits intuitive selection of elements within the object. Scroll bars and elevators are optionally available for feedback. If the device has a pointing mechanism then scroll bars are active and simulate the desktop experience.

#### Personalization and Temporal Adoption

System **100** and **200** permit for the personalization of device **130** by a variety of means. Specifically, what is displayed on screen **137** may depend on either adoption or customization. Adoption refers to the selection of choices, navigation options, etc. are based on user usage patterns. Temporal adoption permits the skins, choices, layouts, content, widgets, etc. to be further influenced by location (for example home, work or traveling) and time of day (including season and day of week). Customization refers to user selectable skins, choices, layouts, dynamic content, widgets, etc. that are available either through a customization on the phone or one that is on the desktop but dynamically linked to the user's other internet connected devices.

To support many personalization functions there must be a convenient method for maintaining, both within a user's session, and between sessions, memory about various user choices and events. Both utilizing a persistent storage mechanism on the device, or a database for user profiles on a server, may be employed.

FIG. 7 shows a display **700** of a mash-up which may be on displayed canvas **305** or on screen **137** of device **130** having the proper Player and Application. Display **700** includes several object **701** that have been dynamically bound, including an indication of time **701a**, an indication of unread text messages **701b**, an RSS news feed **701c** (including 2 "ESPN Top Stories" **701c1** and **701c2**), components **701d** from two Web Services—a weather report ("The Weather Channel"), and a traffic report **701e** ("TRAFFIC.COM").

In assembling the information of display **700**, device **130** is aware of the time and location of the device—in this example the display is for a workday when a user wakes. Device **130** has been customized so that on a work day morning the user wishes to receive the displayed information. Thus in the morning, any messages received overnight would be flagged, the user's favorite RSS sports feeds would be visible, today's weather forecast would be available, and the current traffic conditions between the user's home and office would be graphically depicted. User personalization settings may be

26

maintained as persistent storage on device **130** when appropriate, or in a user profile which is maintained and updated in real-time in a server-side data base.

#### Push Capable Systems

In another embodiment system **100** or **200** is a push-capable system. As an example, of such systems, short codes may be applied to cereal boxes and beverage containers, and SMS text fields can be applied to promotional websites. In either case, a user of device **130** can text the short code or text field to an SMS server, which then serves the appropriate Application link back to device **130**.

FIG. 8 is a schematic of an embodiment of a push enabled system **800**. System **800** is generally similar to system **100** or **200**. Device **130** is shown as part of a schematic of a push capable system **800** in FIG. 8. System **800** includes a website system **801** hosting a website **801**, a server **803** and a content server **805**. System **801** is connected to servers **803** and/or **805** through the Internet. Server **803** is generally similar to server **120**, servers **805** is generally similar to server **140**.

In one embodiment, a user sets up a weekly SMS update from website system **801**. System **801** provides user information to server **803**, which is an SMS server, when an update is ready for delivery. Server **803** provides device **130** with an SMS indication that the subscribed information is available and queries the user to see if they wish to receive the update. Website **801** also provides content server **805** with the content of the update. When a user of device **130** responds to the SMS query, the response is provided to content server **805**, which provides device **130** with updates including the subscribed content.

In an alternative embodiment of system **800**, server **803** broadcasts alerts to one or more devices **130**, such as a logical group of devices. The user is notified in real-time of the pending alert, and can view and interact with the message without interrupting the current Application.

FIG. 9 is a schematic of an alternative embodiment of a push enabled system **900**. System **900** is generally similar to system **100**, **200**, or **800**. In system **900** a user requests information using an SMS code, which is delivered to device **130**. System **900** includes a promotional code **901**, a third-party server **903**, and content server **805**. Server **803** is connected to servers **803** and/or **805** through the Internet, and is generally similar to server **120**.

A promotional code **901** is provided to a user of device **130**, for example and without limitation, on print media, such as on a cereal box. The use of device **130** sends the code server **903**. Server **903** then notifies server **805** to provide certain information to device **130**. Server **805** then provides device **130** with the requested information.

#### Device Routines

Device routines **114** may include, but are not limited to: an authoring tool SDK for custom code development including full set of Java APIs to make it easy to add extensions and functionality to mobile Applications and tie Applications to back-end databases through the content server **140**; an expanding set of web services **250** available through the authoring tool SDK; a web services interface to SOAP/XML enabled web services; and an RSS/Atom and RDF feed collector **1010** and content gateway **1130**.

#### Authoring Tool SDK for Custom Code Development Including Full Set of Java APIs

In one embodiment, authoring platform **110** SDK is compatible for working with various integrated development environments (IDE) and popular plug ins such as J2ME Polish. In one embodiment the SDK would be another plug in to these IDEs. A large and powerful set of APIs and interfaces are thus available through the SDK to permit the seamless

US 9,063,755 B2

27

extension of any Application to back end business logic, web services, etc. These interfaces and APIs may also support listeners and player-side object operations.

There is a large set of listeners that expose both player-side events and dynamically linked server side data base events. Some examples of player side events are: player-side time based event, a page entry event, player-side user interactions and player-side object status. Examples of server-side data base events are when a particular set of linked data base field values change, or some field value exceeds a certain limit, etc.

A superset of all authoring tool functionality is available through APIs for layer-side object operations. These include, but are not limited to: page view level APIs for inserting, replacing, and or modifying any page object; Object Level APIs for modifying any attribute of existing objects, adding definitions to attributes, and adding, hiding or replacing any object.

Authoring Tool SDK Available Web Services

The APIs permit, without limit, respond, with or without relying on back-end business logic, that is, logic that what an enterprise has developed for their business, to any player-side event or server-side dynamically linked data-base, incorporating any open 3rd party web service(s) into the response. RSS/ATOM and RDF Feed Conversion Web Service

FIG. 10 is a schematic of one embodiment a system 1000 having a feed collector 1010. System 1000 is generally similar to system 100, 200, 800, or 900. Feed collector 1010 is a server side component of system 100 that collects RSS, ATOM and RDF format feeds from various sources 1001 and aggregates them into a database 1022 for use by the Applications built using authoring platform 110.

Feed collector 1010 is a standard XML DOM data extraction process, and includes Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, and Custom Populator Rule 1016, DOM XML Parsers 1011, 1015, and 1017, Feed Processed Data Writer 1018, Custom Rule Based Field Extraction 1019, Rule-based Field Extraction 1020, Channel Data Controller 1021, and Database 1022.

The feed collector is primarily driven by two sets of parameters: one is the database schema (written as SQL DDL) which defines the tables in the database, as well as parameters for each of the feeds to be examined. The other is the feed collection rules, written in XML, which can be used to customize the information that is extracted from the feeds. Each of the feeds is collected at intervals specified by the feed parameter set in the SQL DDL.

Feed collector 1010 accepts information from ATOM, RDF or RSS feed sources 1001. Using a rules-based populator, any of these feeds can be logically parsed, with any type of data extraction methodology, either by using supplied rules, or by the author defining their own custom extraction rule. The rules are used by the parser to parse from the feed sources, and the custom rule base field extraction replaces the default rules and assembles the parsed information into the database

In particular, Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, Custom Populator Rule 1016, and DOM XML Parsers 1011, 1015, and 1017 are parse information from the feeds 1001, and Feed Processed Data Writer 1018, Custom Rule Based Field Extraction 1019, Rule-based Field Extraction 1020, and Channel Data Controller 1021, supply the content of the feeds in Database 1022, which is accessible through content server 140.

FIG. 11 is a schematic of an embodiment of a system 1100 having a Mobile Content Gateway 1130. System 1100 is generally similar to system 100, 200, 800, 900, or 1000. System 1100 includes an SDK 1131, feed collector 1010,

28

database listener 1133, transaction server 1134, custom code 1135 generated from the SDK, Java APIs, Web Services 1137, and PDL snippets compacted objects 1139. System 1100 accepts input from Back End Java Code Developer 1120 and SOAP XML from Web Services 1110, and provides dynamic content to server 140 and Players to devices 130.

In one embodiment authoring platform 110 produces a Server-side PDL (SPDL) at authoring time. The SPDL resides in server 120 and provides a logical link between the Application's UI attributes and dynamic content in database 1022. When a user of device 130 requests dynamic information, server 120 uses the SPDL to determine the link required to access the requested content.

In another embodiment Web Services 1137 interface directly with 3rd party Web Services 1110, using SOAP, REST, JAVA, JavaScript, or any other interface for dynamically updating the attributes of the Application's UI objects. XSP Web Pages as a Web Service

In one embodiment, a PDL for a page is embedded within an HTML shell, forming one XSP page. The process of forming XSP includes compressing the description of the page and then embedding the page within an HTML shell.

In another embodiment, a PDL, which contains many individual page definitions, is split into separate library objects on the server, so that each page can be presented as a PDL as part of a Web Service.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java), and The code and data have been reduced by 4 to 10 times.

Compression has two distinct phases. The first takes advantage of how the primitive representations had been assembled, while the second utilizes standard LZ encoding.

The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

One embodiment for compacting data that may be used is described in Rempell. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

In Rempell, a process for compacting a "database" (that is, generating a compact PDL) is described, wherein data objects, including but not limited to, multi media objects such as colors, fonts, images, sound clips, URLs, threads, and video, including multi level animation, transformation, and time line are compacted. As an extension to Rempell in all cases these objects are reduced and transformed to Boolean, integer and string arrays.

The compression technique involves storing data in the smallest arrays necessary to compactly store web page information. The technique also includes an advanced form of delta compression that reduces integers so that they can be stored in a single byte, as high water marks.

Thus, for example, the high water mark for different types of data comprising specific web site settings are stored in a header record as Boolean and integer variables and URL and color objects. Data that defines web page, paragraph, text button, and image style and text button, image and paragraph high watermark settings can be stored in one-dimensional arrays as Boolean, integer and string variables and URL, font, image or thread objects at. The URL, color, font, image and thread objects can also be created as required

Data that defines text button, image, paragraph, or other parent objects and paragraph line high watermark settings can be stored in two-dimensional arrays (by web page and by object number) as Boolean, integer, string, floating point variables and URLs. Again, the URL, color, font, image,

US 9,063,755 B2

29

audio clip, video clip, text area and thread objects can also be created as required. Data that defines a paragraph line and paragraph line segment high watermarks can be stored in three-dimensional arrays (by web page, by paragraph number, and by line number) as Boolean, integer or string variables. Again, the URL, color or font objects can be created as required. Data that defines a paragraph line segment can be stored into four-dimensional arrays (by web page, by paragraph number, by line number and by line number segment) as Boolean, integer or string variables or URL, color and font objects.

As a data field is added, changed or deleted, a determination is made as to whether a value for a given high watermark needs to be changed. If so, it is updated. As a specific method in the build engine is called, a determination is made on whether a feature flag needs to be set. For example, if a particular JAVA method is called, which requires an instance of a certain JAVA Class to be executed by the run time engine, then that JAVA Class is flagged, as well as any supporting methods, variables and/or object definitions.

In one implementation, the header record, the style record, the web page record, and the object records, are carefully defined in a specific order, written in that order, and explicitly cast by object type when read by the run time engine. Exception handling can be implemented to recover from any errors. This helps assure that data integrity is maintained throughout the build and run time processes.

Also described in Rempell is the "run generation process." This is equivalent generating a Player in the present application. This process starts when the build process detects that the user is finished defining the web site (user has saved the web site and invokes the run generation process), and concludes with the actual uploading of all the necessary web site run time files to the user's server.

In one embodiment, the PDL includes a first record, a "Header" record, which contains can include the following information:

- 1: A file format version number, used for upgrading database in future releases.
- 2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user.
- 3: Whether the Application is a web site.
- 4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.
- 5: Web page and styles high watermarks.
- 6: The Websitenam.

As new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the PDL, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

30

The settings for all paragraph, text button and image styles are then written as a style record based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist.

The body of the PDL is then written. All Boolean values are written inside a four-dimensional loop. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects (i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of line segments per paragraph line and contains the values used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment).

All integer values are written inside a four-dimensional loop. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

Multimedia objects are written inside a two-dimensional loop. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a web browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other objects on a web page.

String records are written inside a four-dimensional loop. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The third loop contains values for paragraph line related data, and the inner-most loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized.

Single and double floating-point, and long integer records are written inside a two-dimensional loop. The outer loop may be empty. The inner loop contains mathematical values

US 9,063,755 B2

31

required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

In one embodiment, a versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation.

All external image, video and audio files are resolved. The external references can be copied to designated directories, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries are either installed on the local system or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code. Finally, the run time engine for the web site is created. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file.

Next, an HTML Shell File (HSF) is constructed. The first step of this process is to determine whether the dynamic web page and object resizing is desired by testing the Application setting. If the Application was a web page, and thus requiring dynamic web page and object resizing, virtual screen resolution settings are placed in an appropriate HTML compliant string. If the Application is a banner or other customized Application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values.

An analysis is made for the background definition for the first internal web page. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the web browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a web browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the web browser. More specifically, if the Application required dynamic web page and object resizing then JavaScript and HTML compliant strings are generated so that, when interpreted by the web browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated in order to enable JavaScript to SRS applet communication. In one implementation, the code is generated by performing the following functions:

- 1: Determine the current web browser type.
- 2: Load the SRS from either a JAR or CAB File, based on web browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.

32

5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated that perform the following steps at the time the HSF is initialized by the web browser:

1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.

2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the PDL.

3: Generate an HTML complaint string, dependent upon the type of web browser, which causes the current web browser to load either the JAR or the CAB File(s).

4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the web browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

The writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "Websitename".html file is created.

In one embodiment, the processes for creating the CAB and JAR Files is as follows. The image objects, if any, which were defined on the first internal web page are analyzed. If they are set to draw immediately upon the loading of the first web page, then they are flagged for compression and inclusion in the CAB and JAR Files. The feature flags are analyzed to determine which JAVA classes have been compiled. These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "Websitename".class, customized run time engine file, and the "Websitename".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "Websitename".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "Websitename".bat and "Websitenamelib".bat files are written. The "Websitename".bat and "Websitenamelib".bat files are then executed under DOS, creating compressed "Websitename".cab and "Websitenamelib".cab files and compressed "Websitename".jar and "Websitenamelib".jar files. The HTML Shell File and the JAR and CAB files are then, either as an automatic process, or manually, uploaded to the user's web site. This completes the production of an XSP page that may be accessed through a web browser.

Displaying Content on a Device  
Decompression Management

Authoring platform 110 uses compaction to transform the code and data in an intelligent way while preserving all of the original classes, methods and attributes. This requires both an intelligent server engine and client (handset) Player, both of which fully understand what the data means and how it will be used.

The compaction technology described above includes transformation algorithms that deconstruct the logic and data into their most primitive representations, and then reas-

US 9,063,755 B2

33

sembles them in a way that can be optimally digested by further compression processing. This reassembled set of primitive representations defines the PDL of authoring platform **110**.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java). The data is then compressed by first taking advantage of how the primitive representations had been assembled, and then by utilizing standard LZ encoding. The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

The Player, when preparing a page view for execution, decompresses and then regenerate the original objects, but this time in compliance with the programming APIs of device **130**. Specifically, device **130** operates on compacted image pages, one at a time. The cache manager retrieves, decompresses, and reassembles the compacted page images into device objects, which are then interpreted by device **130** for display on screen **137**.

#### Response Director

In one embodiment, system **100** includes a Response Director, which determines a user's handset, fetches the correct Application from different databases, and delivers a respective highly compressed Application in a PDL format over the air (OTA).

In one embodiment, the Response Director operates on a network connected computer to provide the correct Player to a given device based on the information the device sent to it. As an example, this may occur when a device user enters their phone number into some call-to-action web page. The response director is called and sends an SMS message to the device, which responds, beginning the recognition process.

FIG. 12 illustrates one embodiment of a system **1200** that includes a response director **210**, a user agent database **1201**, an IP address database **1203**, and a file database **1205**. System **1200** is generally similar to system **100**, **200**, **800**, **900**, **1000**, or **1100**.

Databases **1201**, **1203**, and **1205** may reside on server **120**, **210**, or any computer system in communication with response director **210**. System **1200**, any mobile device can be serviced, and the most appropriate Application for the device will be delivered to the device, based on the characteristics of the device.

User agent database **1201** includes user agent information regarding individual devices **130** that are used to identify the operating system on the device. IP address database **1203** identifies the carrier/operator of each device **130**. File database **1205** includes data files that may operate on each device **130**.

The following is an illustrative example of the operation of response director **210**. First, a device **1300** generates an SMS message, which automatically sends an http://stream that includes handset information and its phone number to response director **210**. Response director **210** then looks at a field in the http header (which includes the user agent and IP address) that identifies the web browser (i.e., the "User Agent"). The User Agent prompts a database lookup in user agent database **1201** which returns data including, but not limited to, make, model, attributes, MIDP 1.0 MIDP 2.0, WAP and distinguishes the same models from different countries. A lookup of the IP address in IP address **1203** identifies the carrier/operator.

File database **1205** contains data types, which may include as jad1, jad2, html, wml/wap2, or other data types, appropriate for each device **130**. A list of available Applications are returned to a decision tree, which then returns, to device **130**, the Application that is appropriate for the respective device.

34

For each file type, there is an attributes list (e.g., streaming video, embedded video, streaming audio, etc.) to provide enough information to determine what to send to the handset.

Response director **210** generates or updates an html or jad file populating this text file with the necessary device and network dependent parameters, including the Application dependent parameters, and then generate, for example, a CAB or JAD file which contains the necessary Player for that device. For example, the jad file could contain the operator or device type or extended device-specific functions that the player would then become aware of.

If there is an Application that has a data type that device **130** cannot support, for example, video, response director **210** sends an alternative Application to the handset, for example one that has a slide show instead. If the device cannot support a slide show, an Application might have text and images and display a message that indicates it does not support video.

Another powerful feature of response director **210** is its exposed API from the decision tree that permits the overriding of the default output of the decision tree by solution providers. These solution providers are often licensees who want to further refine the fulfillment of Applications and Players to specific devices beyond what the default algorithms provide. Solution providers may be given a choice of Applications and then can decide to use the defaults or force other Applications.

Authoring platform **110** automatically scales Applications at publishing time to various form factors to reduce the amount of fragmentation among devices, and the Response Director serves the appropriately scaled version to the device. For example, a QVGA Application will automatically scale to the QCIF form factor. This is important because one of the most visible forms of fragmentation resides in the various form factors of wireless, and particularly mobile, devices, which range from 128x128, 176x208, 240x260, 220x220, and many other customized sizes in between.

FIG. 13 is a schematic of an embodiment of a system **1300**. System **1300** is generally similar to system **1200**. System **1300** is an overview of the entire Player fulfillment process, starting with the generation of players during the player build process.

System **1300** includes response director **210**, a device characteristics operator and local database **1301**, a player profile database **1303** and a player build process **1305**, which may be authoring platform **110**.

As an example of system **1300**, when response director **210** receives an SMS message from device **130**, the response director identifies the device characteristics operator and locale from database **1301** and a Player URL from database **1303** and provides the appropriate Player to the device.

In another embodiment, Player P extend the power of response director **210** by adapting the Application to the resources and limitations of any particular device. Some of these areas of adaptation include the speed of the device's microprocessor, the presence of device resources such as cameras and touch screens. Another area of adaptation is directed to heap, record store and file system memory constraints. In one embodiment, the Player will automatically throttle down an animation to the frame rate that the device can handle so that the best possible user experience is preserved. Other extensions include device specific facilities such as location awareness, advanced touch screen interactions, push extensions, access to advanced phone facilities, and many others

#### Memory Management

In one embodiment, Player P includes a logical page virtual memory manager. This architecture requires no supporting

US 9,063,755 B2

35

hardware and works efficiently with constrained devices. All page view images, which could span multiple Applications, are placed in a table as highly compacted and compressed code. A typical page view will range from 500 bytes up to about 1,500 bytes. (See, for example, the Rempell patent) When rolled into the heap and instantiated this code increases to the more typical 50,000 up to 250,000 bytes. Additional alert pages may also be rolled into the heap and superimposed on the current page view. Any changes to any page currently downloaded are placed in a highly compact change vector for each page, and rolled out when the page is discarded. Note that whenever an Application is visited that had previously been placed in virtual memory the Server is interrogated to see if a more current version is available, and, if so, downloads it. This means that Application logic can be changed in real-time and the results immediately available to mobile devices.

To operate efficiently with the bandwidth constraints of mobile devices, authoring platform 110 may also utilize anticipatory streaming and multi-level caching. Anticipatory streaming includes multiple asynchronous threads and IO request queues. In this process, the current Application is scanned to determine if there is content that is likely to be required in as-yet untouched page views. Anticipatory streaming also looks for mapping Applications, where the user may zoom or pan next so that map content is retrieved prior to the user requesting it. For mapping applications, anticipatory streaming downloads a map whose size is greater than the map portal size on the device and centered within the portal. Any pan operation will anticipatory stream a section of the map to extend the view in the direction of the pan while, as a lower priority, bring down the next and prior zoom levels for this new geography. Zooming will always anticipatory stream the next zoom level up and down.

Multi-level caching determines the handset's heap through an API, and also looks at the record store to see how much memory is resident. This content is placed in record store and/or the file system, and may, if there is available heap, also place the content there as well. Multi-level caching permits the management of memory such that mobile systems best use limited memory resources. Multi-level caching is a memory management system with results similar to embedding, without the overhead of instantiating the content. In other words, with multi-level caching, handset users get an "embedded" performance without the embedded download. Note that when content is flagged as cacheable and is placed in persistent storage, a digital rights management (DRM) solution will be used.

One embodiment of each of the methods described herein is in the form of a computer program that executes on a processing system. Thus, as will be appreciated by those skilled in the art, embodiments of the present invention may be embodied as a method, an apparatus such as a special purpose apparatus, an apparatus such as a data processing system, or a carrier medium, e.g., a computer program product. The carrier medium carries one or more computer readable code segments for controlling a processing system to implement a method. Accordingly, aspects of the present invention may take the form of a method, an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. Furthermore, the present invention may take the form of carrier medium (e.g., a computer program product on a computer-readable storage medium) carrying computer-readable program code segments embodied in the medium. Any suitable computer readable medium may be used including a mag-

36

netic storage device such as a diskette or a hard disk, or an optical storage device such as a CD-ROM.

It will be understood that the steps of methods discussed are performed in one embodiment by an appropriate processor (or processors) of a processing (i.e., computer) system executing instructions (code segments) stored in storage. It will also be understood that the invention is not limited to any particular implementation or programming technique and that the invention may be implemented using any appropriate techniques for implementing the functionality described herein. The invention is not limited to any particular programming language or operating system. It should thus be appreciated that although the coding for programming devices has not been discussed in detail, the invention is not limited to a specific coding method. Furthermore, the invention is not limited to any one type of network architecture and method of encapsulation, and thus may be utilized in conjunction with one or a combination of other network architectures/protocols.

Reference throughout this specification to "one embodiment," "an embodiment," or "certain embodiments" means that a particular feature, structure or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, appearances of the phrases "in one embodiment," "in an embodiment," or "in certain embodiments" in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures or characteristics may be combined in any suitable manner, as would be apparent to one of ordinary skill in the art from this disclosure, in one or more embodiments.

Throughout this specification, the term "comprising" shall be synonymous with "including," "containing," or "characterized by," is inclusive or open-ended and does not exclude additional, unrecited elements or method steps. "Comprising" is a term of art which means that the named elements are essential, but other elements may be added and still form a construct within the scope of the statement. "Comprising" leaves open for the inclusion of unspecified ingredients even in major amounts.

Similarly, it should be appreciated that in the above description of exemplary embodiments, various features of the invention are sometimes grouped together in a single embodiment, figure, or description thereof for the purpose of streamlining the disclosure and aiding in the understanding of one or more of the various inventive aspects. This method of disclosure, however, is not to be interpreted as reflecting an intention that the claimed invention requires more features than are expressly recited in each claim. Rather, as the following claims reflect, inventive aspects lie in less than all features of a single foregoing disclosed embodiment, and the invention may include any of the different combinations embodied herein. Thus, the following claims are hereby expressly incorporated into this Mode(s) for Carrying Out the Invention, with each claim standing on its own as a separate embodiment of this invention.

Thus, while there has been described what is believed to be the preferred embodiments of the invention, those skilled in the art will recognize that other and further modifications may be made thereto without departing from the spirit of the invention, and it is intended to claim all such changes and modifications as fall within the scope of the invention. For example, any formulas given above are merely representative of procedures that may be used. Functionality may be added or deleted from the block diagrams and operations may be



US 9,063,755 B2

37

interchanged among functional blocks. Steps may be added or deleted to methods described within the scope of the present invention.

We claim:

1. A system for generating code to provide content on a display of a device, said system comprising:

computer memory storing a registry of:

- a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, and
- b) the address of the web service;

an authoring tool configured to:

define a user interface (UI) object for presentation on the display, where said UI object corresponds to the web component included in said registry selected from the group consisting of an input of the web service and an output of the web service,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object,

produce an Application including the selected symbolic name of the defined UI object, where said Application is a device-independent code, and

produce a Player, where said Player is a device-dependent code;

such that, when the Application and Player are provided to the device and executed on the device, and when a user of the device provides one or more input values associated with an input symbolic name to an input of defined UI object,

1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,

2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,

3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for a display of the device to present an output value in the defined UI object.

2. The system of claim 1, where said registry includes definitions of input and output related to said web service.

3. The system of claim 1, where said web component is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

4. The system of claim 1, where said UI object is an input field for a chat.

5. The system of claim 1, where said UI object is an input field for a web service.

6. The system of claim 1, where said UI object is an input field usable to obtain said web component, where said input field includes a text field, a scrolling text box, a check box, a drop down-menu, a list menu, or a submit button.

7. The system of claim 1, where said web component is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

8. The system of claim 1, where said authoring tool is further configured to:

define a phone field or list; and

38

generate code that, when executed on the device, allows a user to supply a phone number to said phone field or list.

9. The system of claim 1, where said authoring tool is further configured to:

define a SMS field or list; and

generate code that, when executed on the device, allows a user to supply an SMS address to said SMS field or list.

10. The system of claim 1,

where said code includes three or more codes, where one of said three or more codes is device specific, and where two of said three or more codes is device independent.

11. The system of claim 1, where said code is provided over said network.

12. A method of displaying content on a display of a device utilizing a registry of one or more web components related to inputs and outputs of a web service obtainable over a network, where each web component includes a plurality of symbolic names of inputs and outputs associated with each web service, and where the registry includes: a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, and

b) the address of the web service, said method comprising: defining a user interface (UI) object for presentation on the display, where said UI object corresponds to a web component included in said registry selected from the group consisting of an input of the web service and an output of the web service;

selecting a symbolic name from said web component corresponding to the defined UI object;

associating the selected symbolic name with the defined UI object;

producing an Application including the selected symbolic name of the defined UI object, where said Application is a device-dependent code; and

producing a Player, where said Player is a device-dependent code;

such that, when the Application and Player are provided to the device and executed on the device, and when a user of the device provides one or more input values associated with an input symbolic name to an input of defined UI object,

1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,

2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,

3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for a display of the device to present an output value in the defined UI object.

13. The method of claim 12, where said registry includes definitions of input and output related to said web service.

14. The method of claim 12, where said web component is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

15. The method of claim 12, where said UI object is an input field for a chat.

16. The method of claim 12, where said UI object is an input field for a web service.

17. The method of claim 12, where said UI object is an input field usable to obtain said web component, where said

US 9,063,755 B2

39

input field includes a text field, a scrolling text box, a check box, a drop down-menu, a list menu, or a submit button.

18. The method of claim 12, where said web component is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

19. The method of claim 12, further comprising:

defining a phone field or list; and

generating code that, when executed on the device, allows a user to supply a phone number to said phone field or list.

20. The method of claim 12, further comprising:

defining a SMS field or list; and

generating code that, when executed on the device, allows a user to supply an SMS address to said SMS field or list.

21. Previously Presented The method of claim 12, and such that said Player interprets dynamically received, device-independent values of the web component defined in the Application.

22. The method of claim 12, further comprising:

providing said Application and Player over said network.

23. A method of providing information to a device having a display from a web component of a web service to a device on a network, said method comprising:

accepting, on the device, a first code over the network, where said first code is device-dependent;

accepting, on the device, a second code over the network, where said second code is device-independent and includes a plurality of symbolic names of inputs and outputs associated with the web service; and

executing said first code on the device,

where the symbolic names are provided from a registry of one or more web components related to inputs and outputs of a web service obtainable over a network,

40

where the web service requires both an input symbolic name and one or more associated input values and returns one or more output values having an associated output symbolic name, and

where the registry includes

a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, and

b) the address of the web service;

where said executing includes:

processing said symbolic names of the second code on the device,

transmitting processed instructions from the device to the web service, and

accepting a third code on the device over the network, where said third code is a device-independent third code including the output of the web component provided by the web service over the network and in response to the second code.

24. The method of claim 23, where said third code is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

25. The method of claim 23, where said third code is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

26. The method of claim 23, where said first code and said second code are generated using an authoring tool.

27. The method of claim 23, where said first code is a Player.

28. The method of claim 23, where said second code is an Application which includes one or more web components.

\* \* \* \* \*

UNITED STATES PATENT AND TRADEMARK OFFICE  
**CERTIFICATE OF CORRECTION**

PATENT NO. : 9,063,755 B2  
APPLICATION NO. : 12/936395  
DATED : June 23, 2015  
INVENTOR(S) : Rempell et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page, item 73, Assignee: Express Mobile, inc., Novato, CA (US) - the "inc." should be -- Inc. --.

In the claims

Column 39, line 18, Claim 21, delete the text reading "Previously Presented".

Signed and Sealed this  
Twentieth Day of October, 2015



Michelle K. Lee  
*Director of the United States Patent and Trademark Office*



US009471287B2

(12) **United States Patent**  
**Rempell et al.**

(10) **Patent No.:** **US 9,471,287 B2**  
(45) **Date of Patent:** **\*Oct. 18, 2016**

(54) **SYSTEMS AND METHODS FOR  
INTEGRATING WIDGETS ON MOBILE  
DEVICES**

(71) Applicant: **Express Mobile, Inc.**, Novato, CA (US)

(72) Inventors: **Steven H. Rempell**, Novato, CA (US);  
**David Chrobak**, Clayton, CA (US);  
**Ken Brown**, San Martin, CA (US)

(73) Assignee: **Express Mobile, Inc.**, Novato, CA (US)

(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **14/708,074**

(22) Filed: **May 8, 2015**

(65) **Prior Publication Data**

US 2015/0317130 A1 Nov. 5, 2015

**Related U.S. Application Data**

(63) Continuation of application No. 12/936,395, filed as application No. PCT/US2009/039695 on Apr. 6, 2009, now Pat. No. 9,063,755.

(60) Provisional application No. 61/123,438, filed on Apr. 7, 2008, provisional application No. 61/113,471, filed on Nov. 11, 2008, provisional application No. 61/166,651, filed on Apr. 3, 2009.

(51) **Int. Cl.**  
**G06F 3/048** (2013.01)  
**G06F 9/44** (2006.01)  
**H04L 29/08** (2006.01)  
**G06F 3/0484** (2013.01)  
**G06F 3/0482** (2013.01)  
**H04L 29/06** (2006.01)  
**H04L 12/58** (2006.01)

(52) **U.S. Cl.**

CPC ..... **G06F 8/34** (2013.01); **G06F 3/0482** (2013.01); **G06F 3/04842** (2013.01); **G06F 9/4443** (2013.01); **H04L 51/046** (2013.01); **H04L 65/60** (2013.01); **H04L 67/02** (2013.01)

(58) **Field of Classification Search**

CPC ..... **G06F 3/048**  
See application file for complete search history.

(56) **References Cited**

**U.S. PATENT DOCUMENTS**

2004/0055017 A1 3/2004 Delpuch et al.  
2004/0163020 A1 8/2004 Sidman  
2004/0199614 A1\* 10/2004 Shenfield ..... H04L 29/06  
709/220

(Continued)

**OTHER PUBLICATIONS**

Stina Nylander et al. "The Ubiquitous Interactor—Device Independent Access to Mobile Services" (Computer-Aided Design for User Interfaces IV, Proceedings of the Fifth International Conference on Computer-Aided Design of User Interfaces CADUI'2004, Jan. 2004, pp. 271-282).\*

(Continued)

*Primary Examiner* — Jennifer To

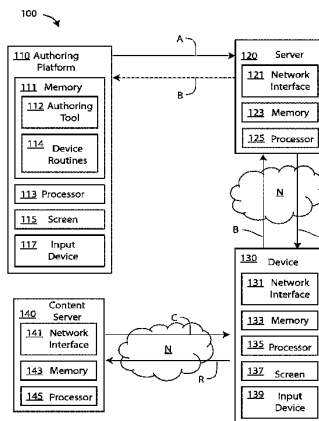
*Assistant Examiner* — Xuyang Xia

(74) *Attorney, Agent, or Firm* — Steven R. Vosen

(57) **ABSTRACT**

Embodiments of a system and method are described for generating and distributing programming to mobile devices over a network. Devices are provided with Players specific to each device and Applications that are device independent. Embodiments include a full-featured WYSIWYG authoring environment, including the ability to bind web components to objects.

**28 Claims, 18 Drawing Sheets**



**US 9,471,287 B2**

Page 2

(56)

**References Cited**

**U.S. PATENT DOCUMENTS**

2005/0149935 A1 7/2005 Benedetti  
 2005/0273705 A1\* 12/2005 McCain ..... G06F 17/24  
 2006/0063518 A1 3/2006 Paddon et al. 715/234

**OTHER PUBLICATIONS**

Stina Nylander et al. "The Ubiquitous Interactor—Device Independent Access to Mobile Services" (Computer-Aided Design for User Interfaces IV, Proceedings of the Fifth international Conference on Computer-Aided Design of User Interfaces CADUT2004, Jan. 2004, pp. 271-282).

International Search Report and Written Opinion —PCT/US2009/039695—Aug. 21, 2009.

International Preliminary Report on Patentability and Written Opinion—PCT/US2009/039695—Oct. 21, 2010.

Rempell et al, co-pending U.S. Appl. No. 14/708,087, filed May 8, 2015.

Rempell et al, co-pending U.S. Appl. No. 14/708,094, filed May 8, 2015.

Rempell et al, co-pending U.S. Appl. No. 14/708,097, filed May 8, 2015.

Rempell et al, co-pending U.S. Appl. No. 14/708,100, filed May 8, 2015.

Rempell et al, co-pending U.S. Appl. No. 14/708,108, filed May 8, 2015.

\* cited by examiner

U.S. Patent

Oct. 18, 2016

Sheet 1 of 18

US 9,471,287 B2

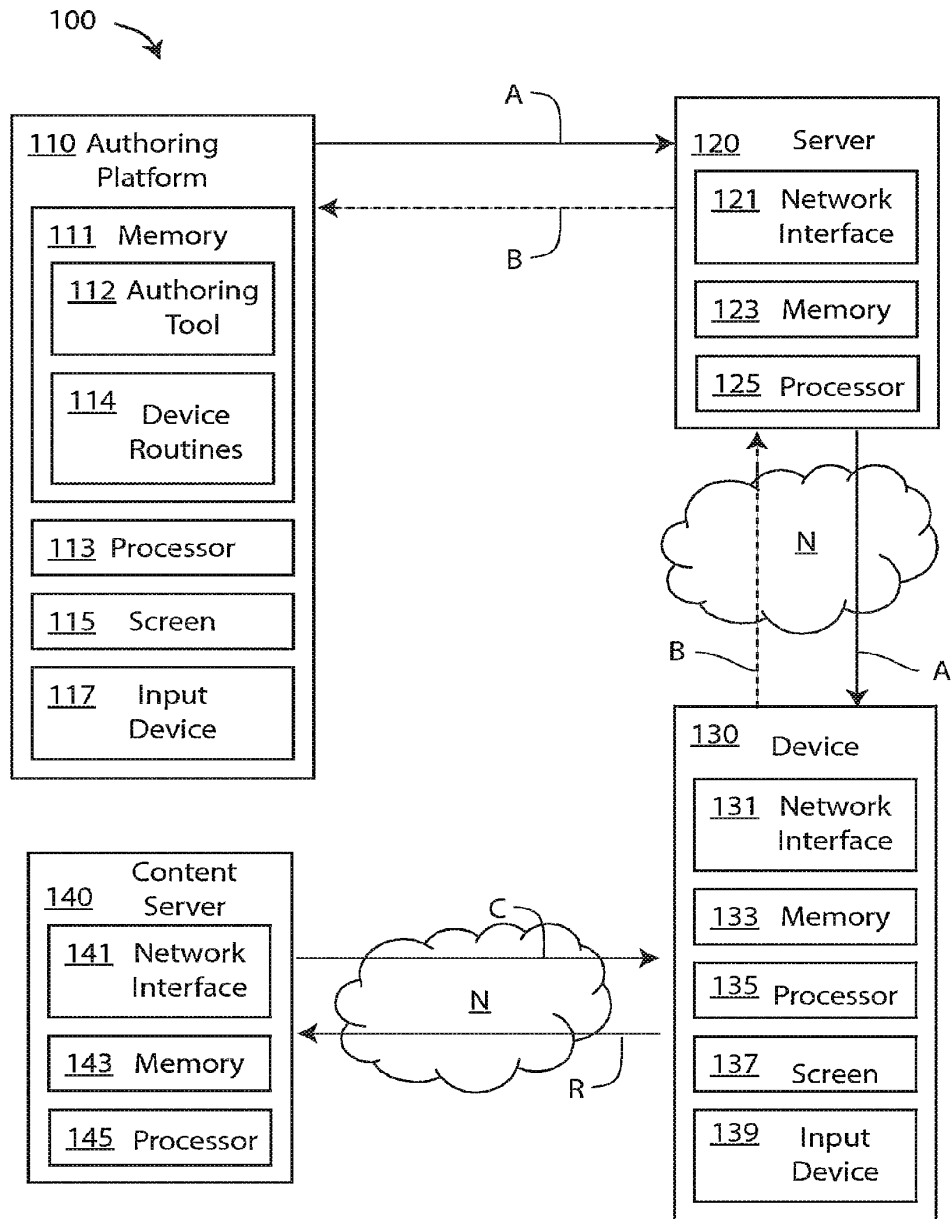


FIG. 1A

U.S. Patent

Oct. 18, 2016

Sheet 2 of 18

US 9,471,287 B2

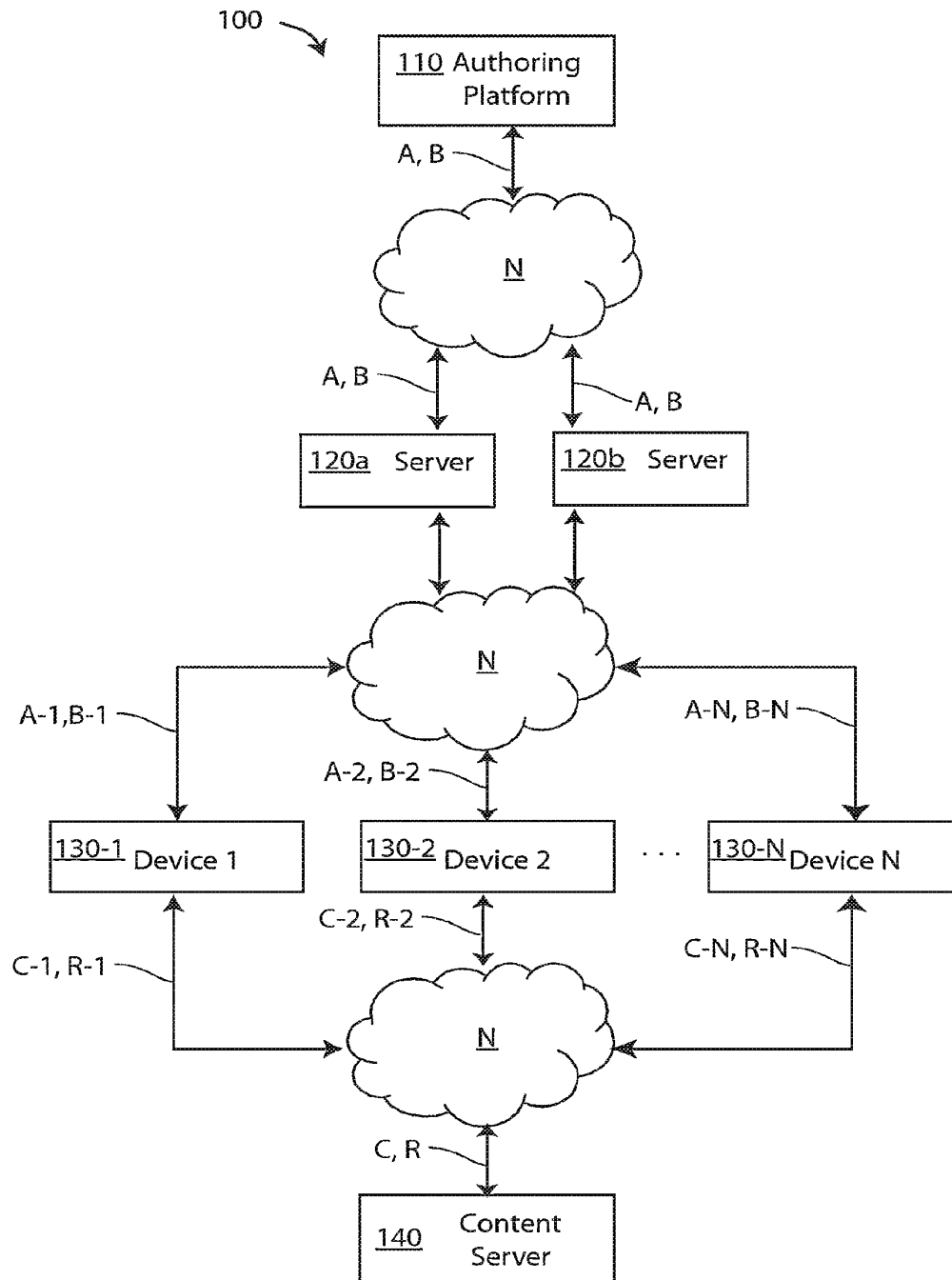


FIG. 1B

U.S. Patent

Oct. 18, 2016

Sheet 3 of 18

US 9,471,287 B2

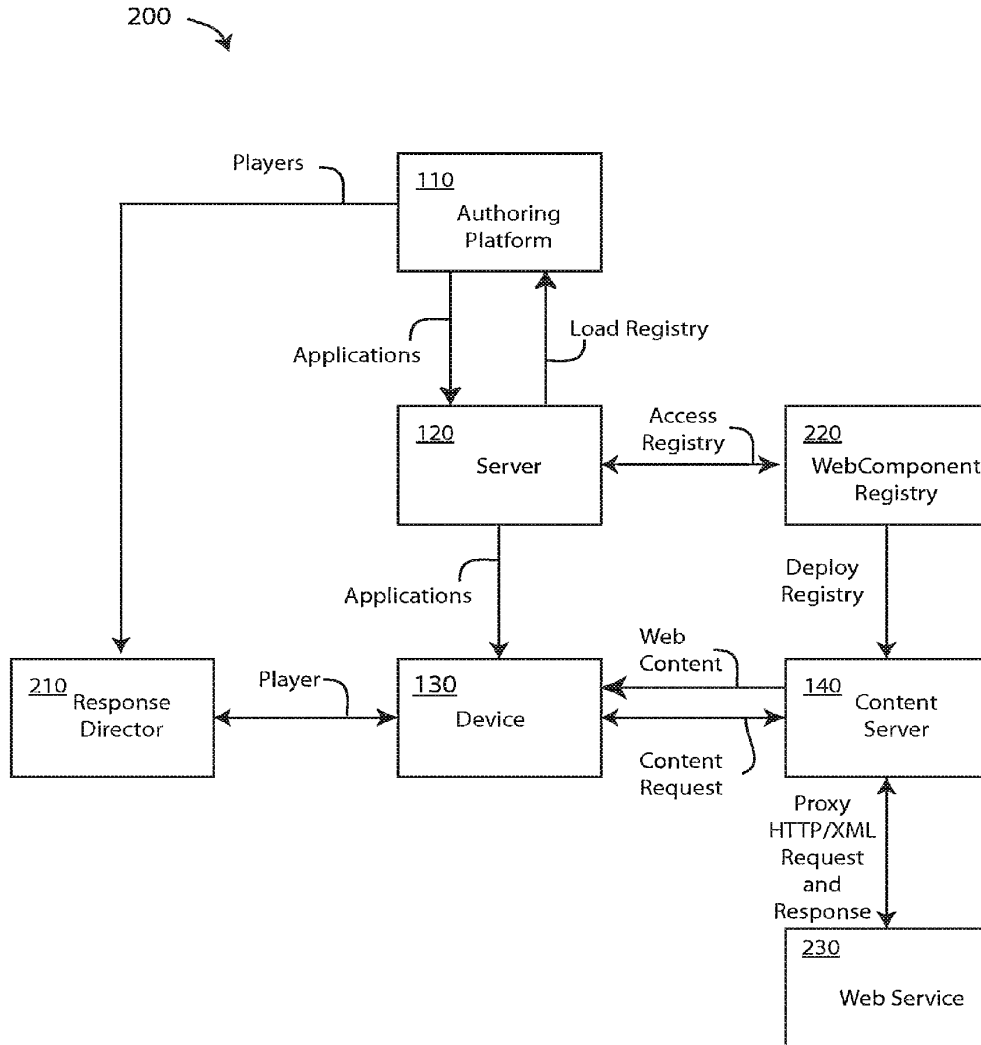


FIG. 2A



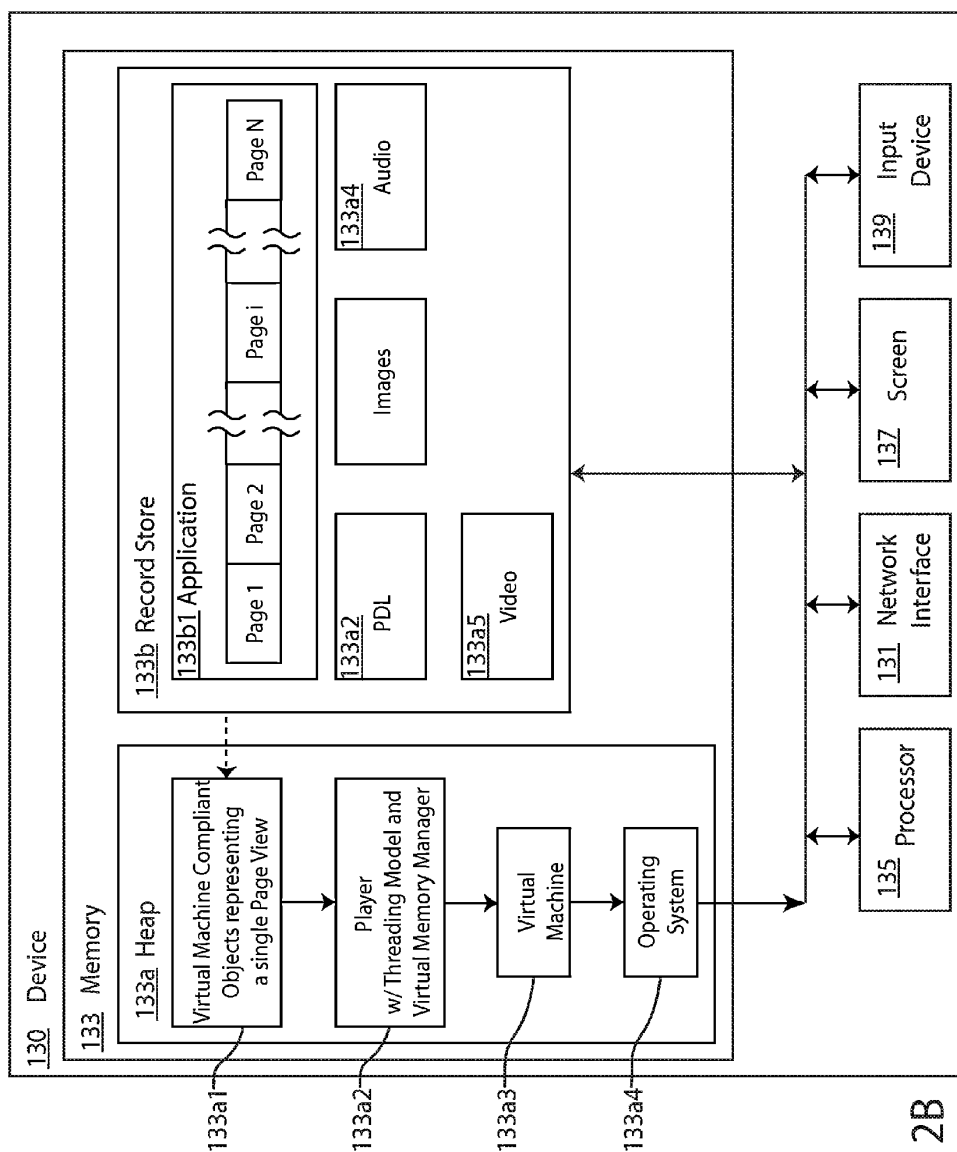
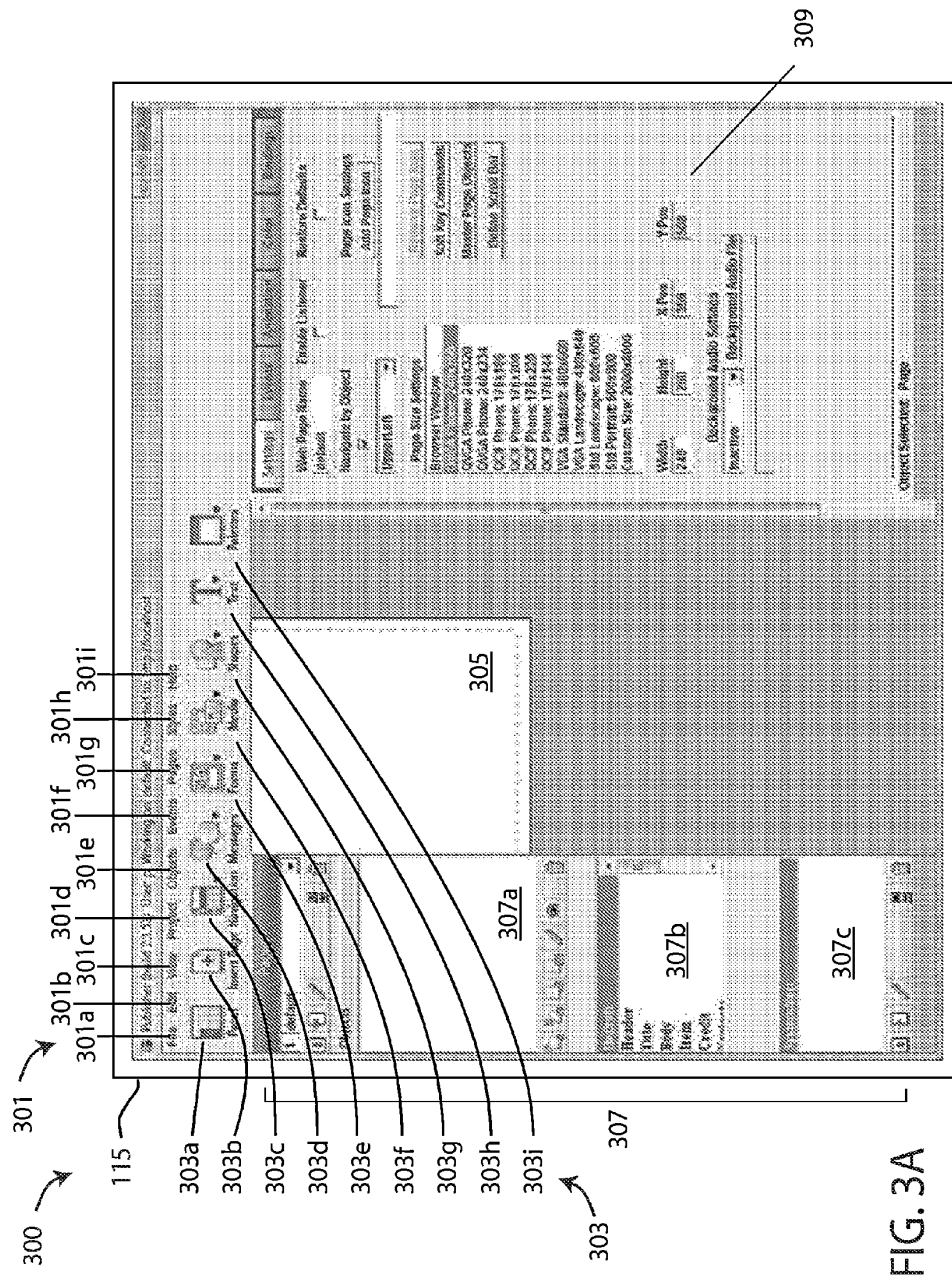


FIG. 2B



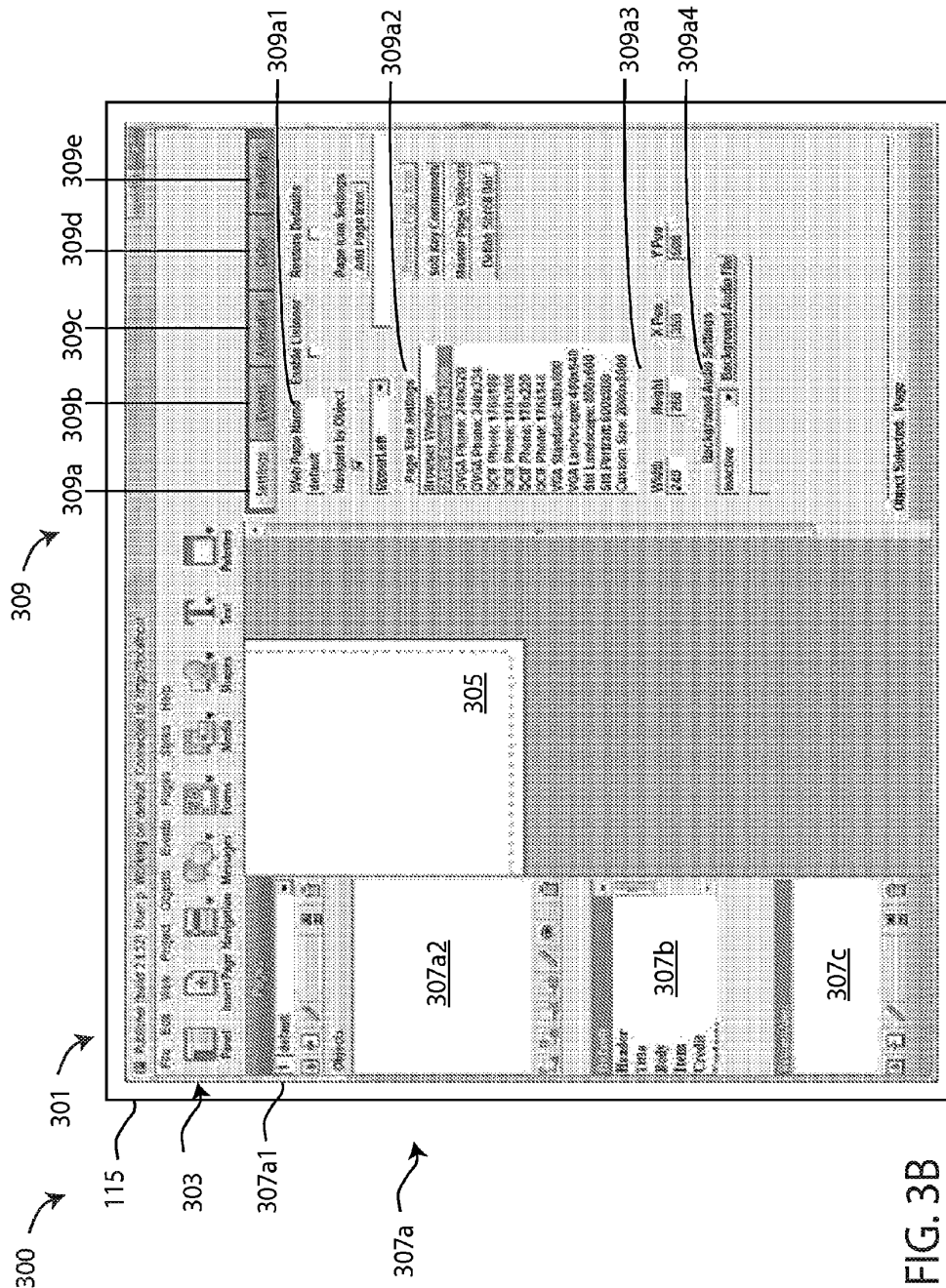


FIG. 3B

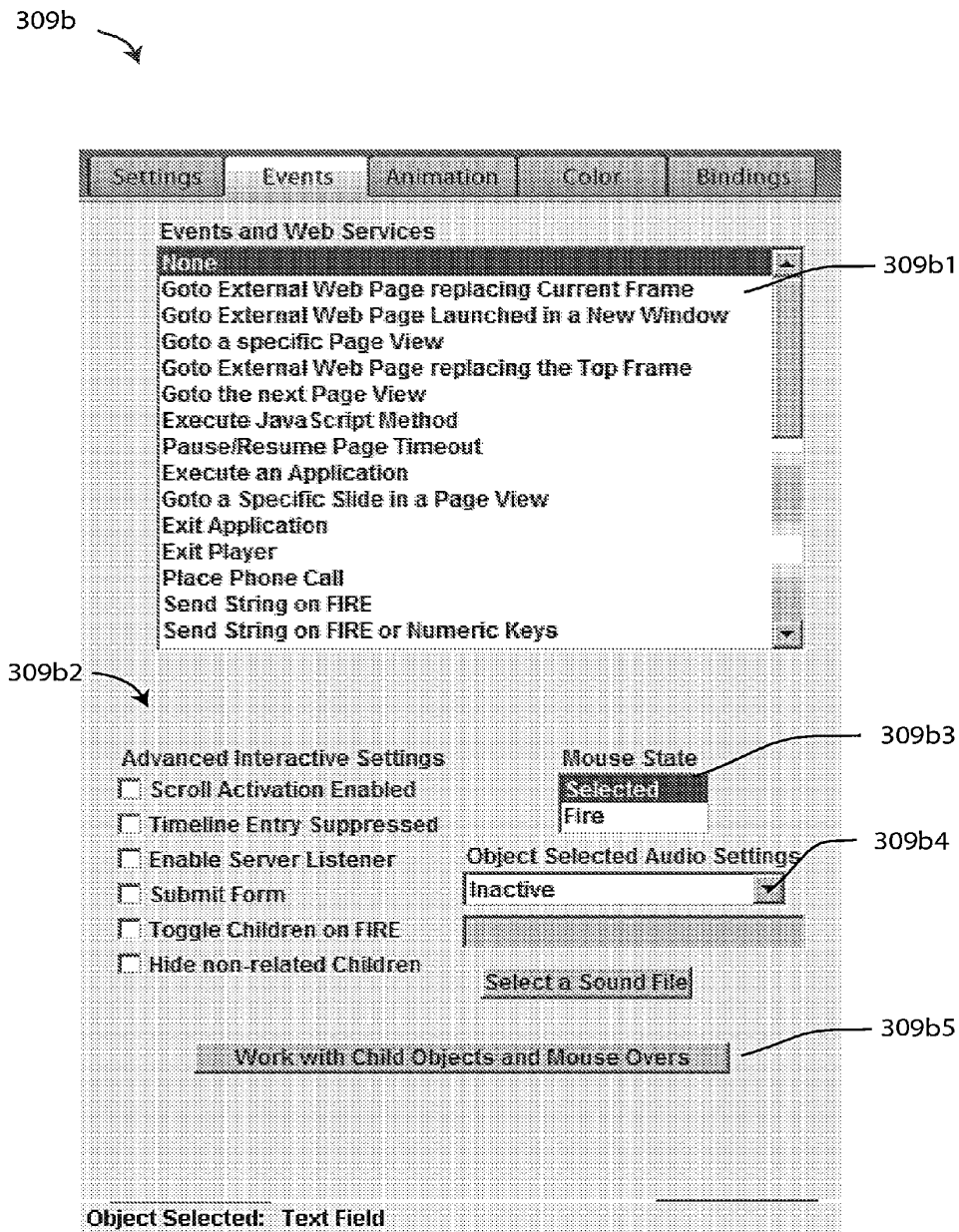


FIG. 3C

U.S. Patent

Oct. 18, 2016

Sheet 8 of 18

US 9,471,287 B2

309c

Settings Events Animation Color Bindings

**Activate Timeline** Object Entry Timeline Specifications

Delay (Sec) Direction Movement Duration (Sec) Frames

☐ 0 .0 None None 2 .0 10

Specifications for this Object's Entry Animation Audio Track

Inactive Entry Audio File

**Object Animation Specifications**

Delay (Sec)	Direction	Movement	Duration (Sec)	Frames
0	None	None	0	1
1	Scroll Left	Fade	1	2
2	Scroll Right	Fade In	2	3
3	Custom	Fade Out	3	4
4	Multi-Point		4	5
5	Seek Cursor		5	6
6	Attach		6	7
7	Deposit		7	8
8	Send Home		8	9
9	Carom N		9	10
10	Carom NE		10	11

Pathname for this Object's Animation Audio Track

Inactive Main Audio File

Animation Cycles Custom Zoom % Avoid Cursor Dampen Anim.

1 0 ☒ ☐

**Object Exit Timeline Specifications**

Activate Delay (Sec) Direction Movement Duration (Sec) Frames

☐ 0 .0 None None 2 .0 10

Specifications for this Object's Exit Animation Audio Track

Inactive Exit Audio File

FIG. 3D

U.S. Patent

Oct. 18, 2016

Sheet 9 of 18

US 9,471,287 B2

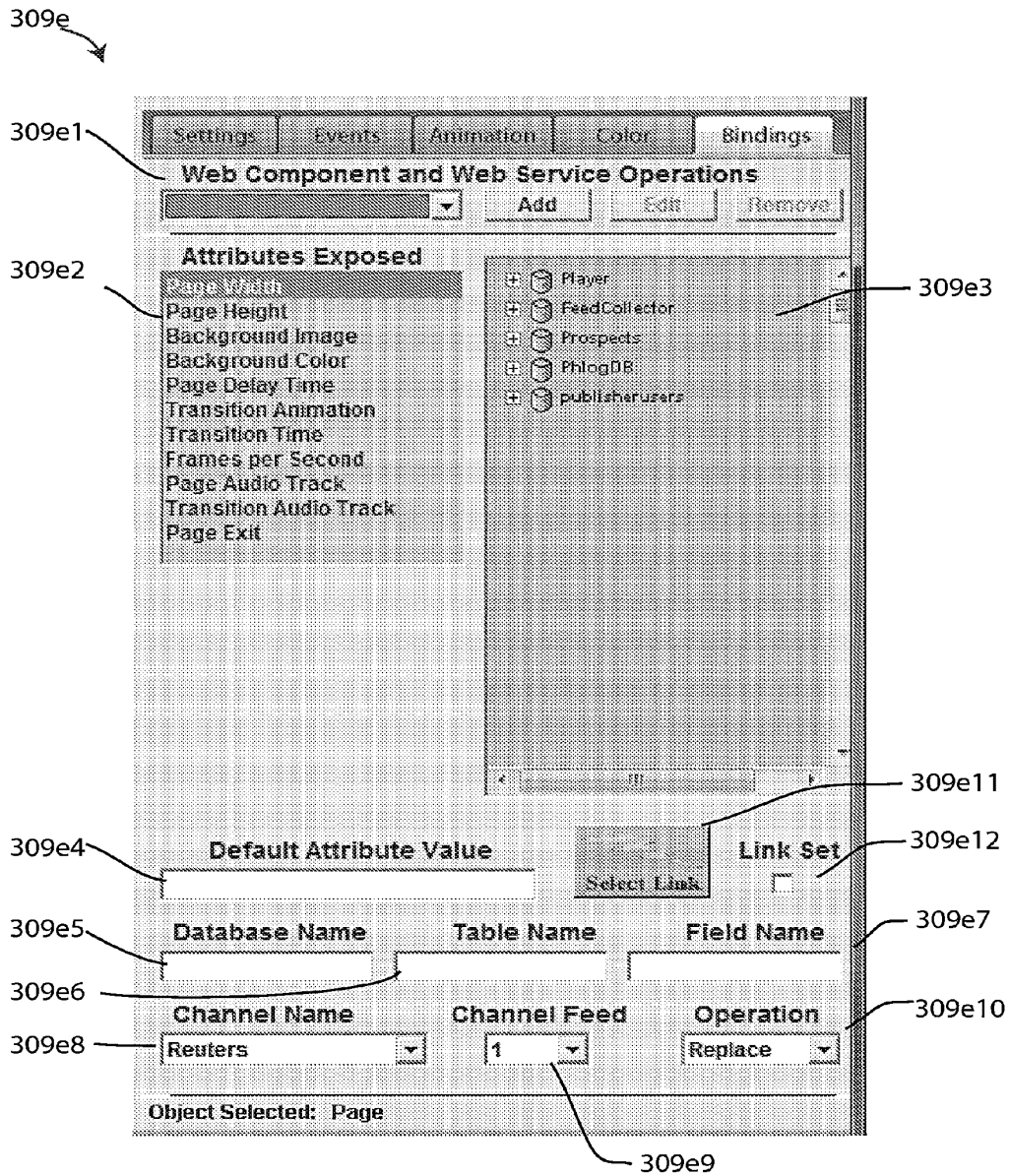


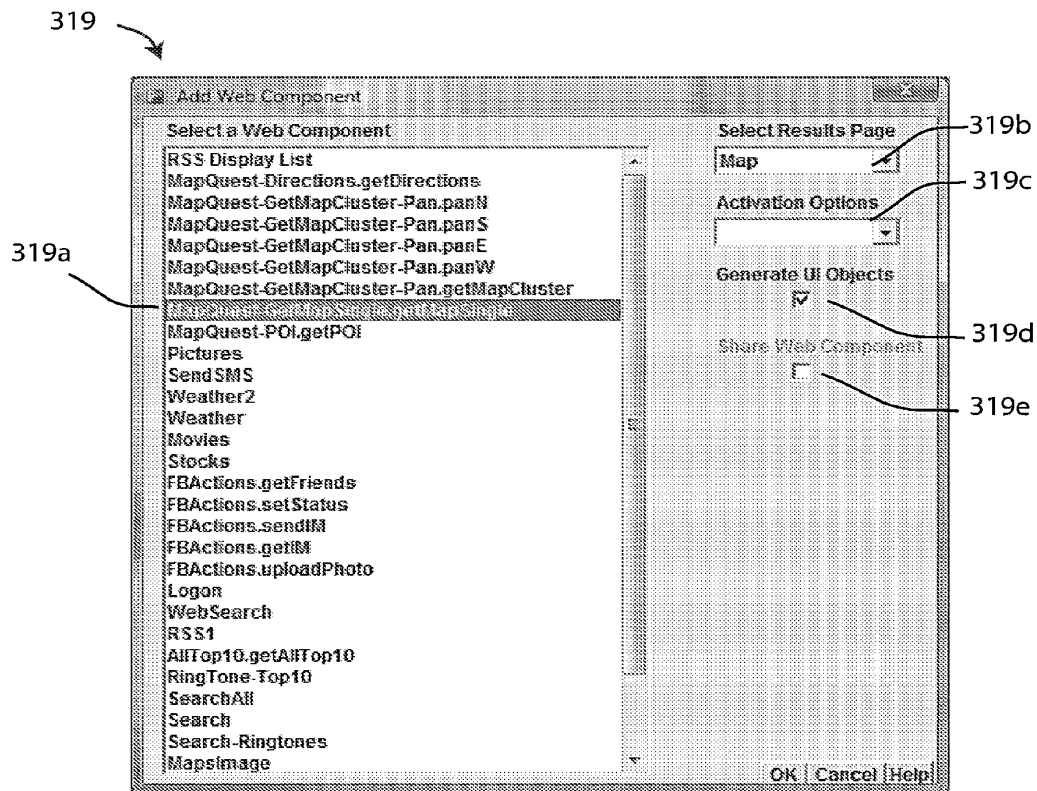
FIG. 3E

U.S. Patent

Oct. 18, 2016

Sheet 10 of 18

US 9,471,287 B2



U.S. Patent

Oct. 18, 2016

Sheet 11 of 18

US 9,471,287 B2

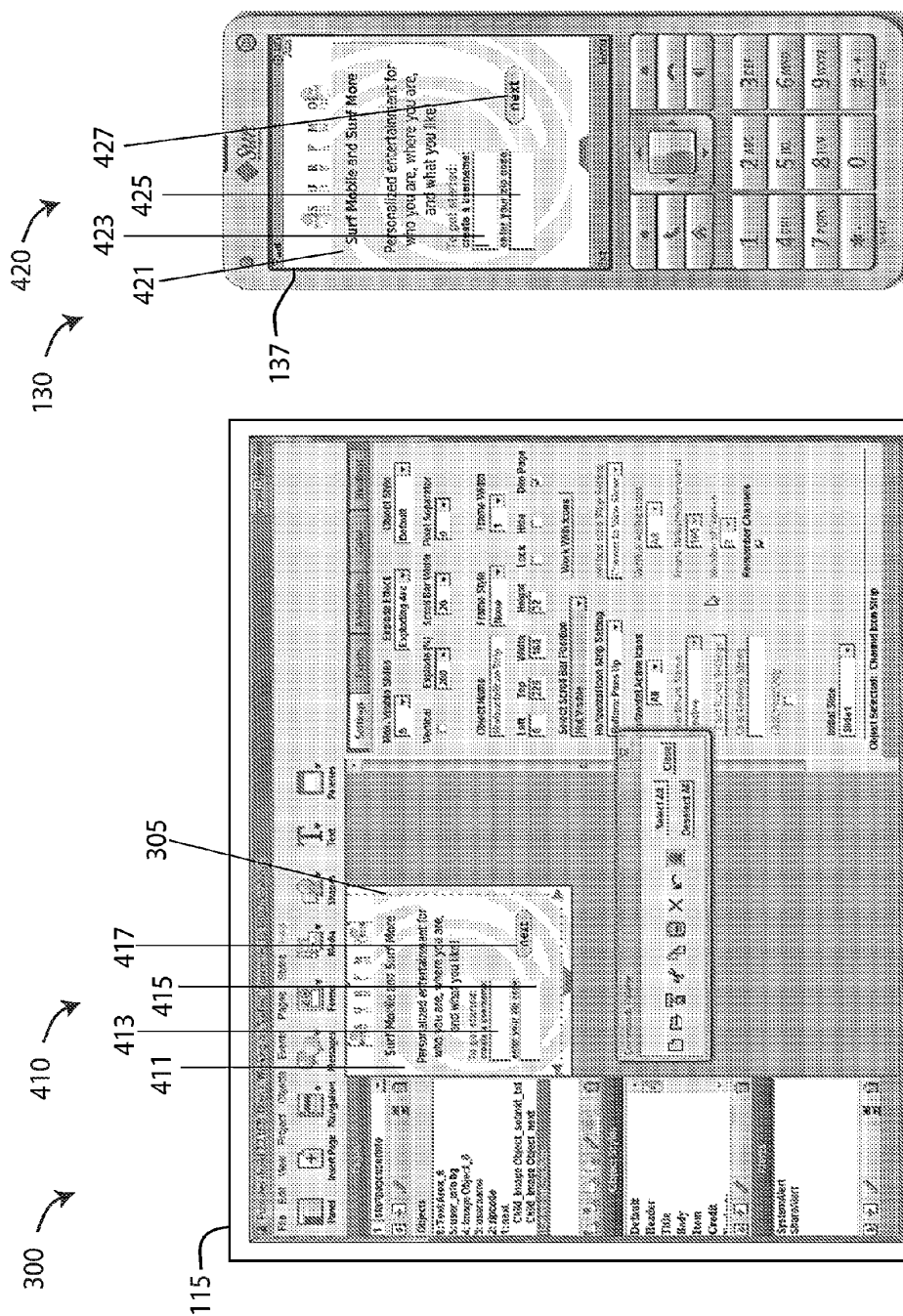
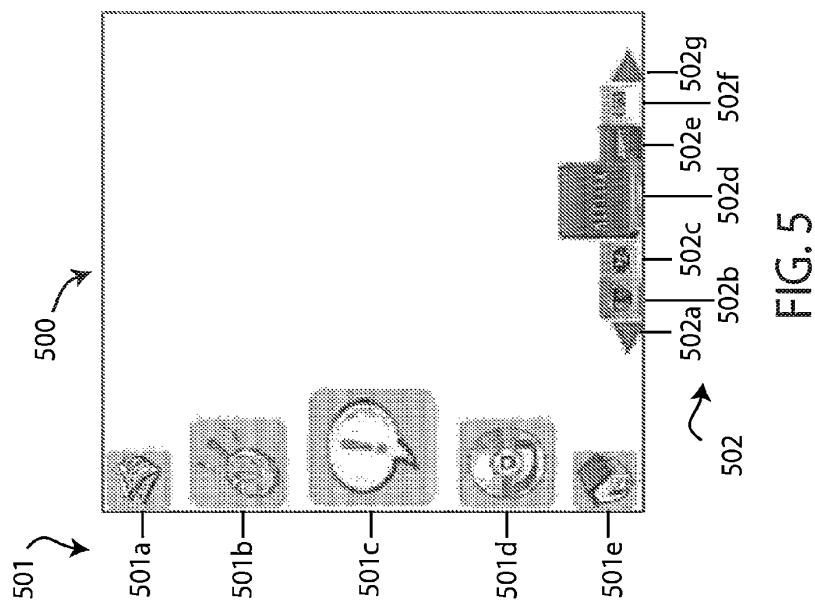
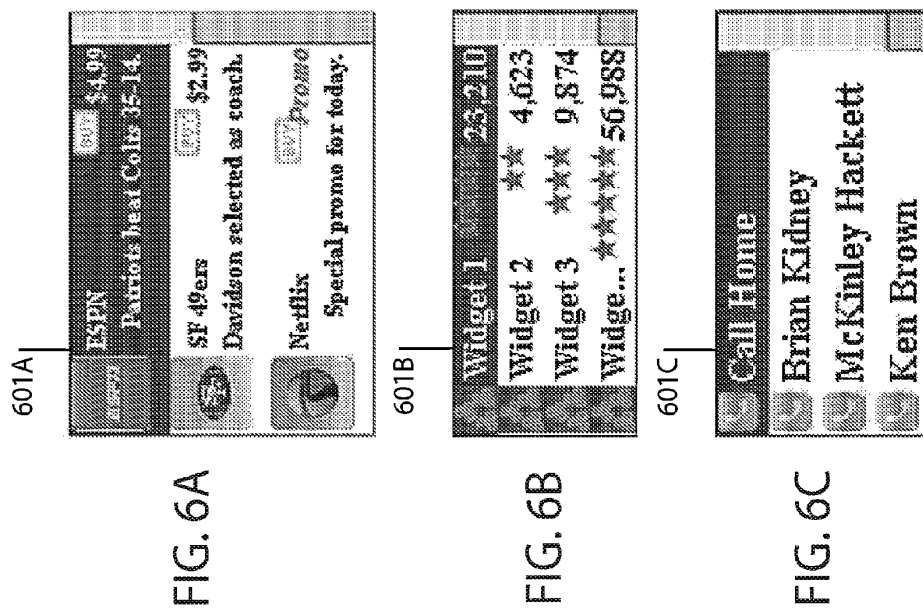


FIG. 4B

FIG. 4A





U.S. Patent

Oct. 18, 2016

Sheet 13 of 18

US 9,471,287 B2

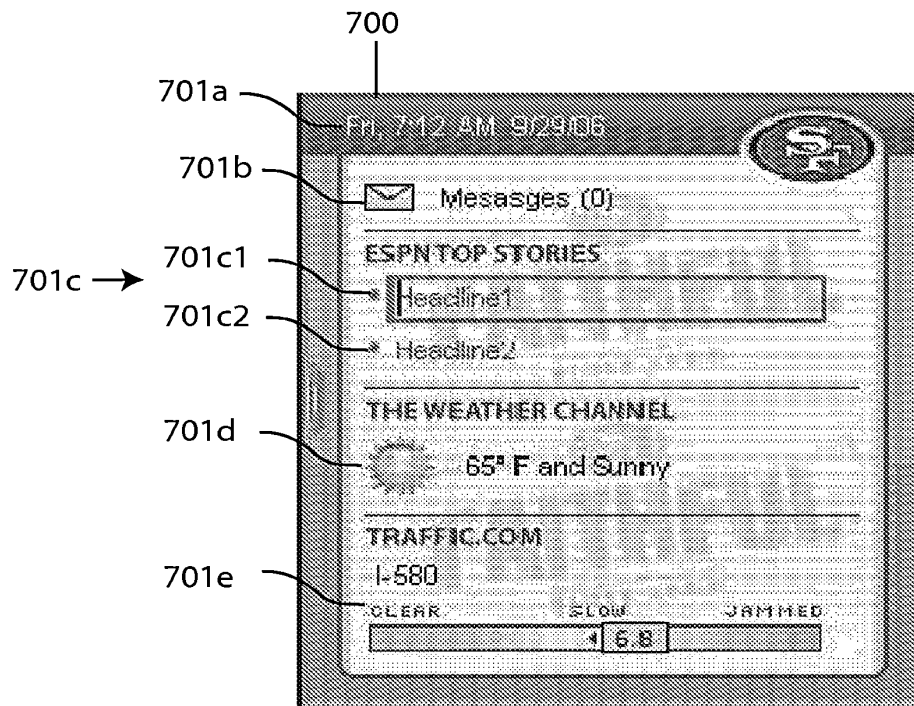


FIG. 7

U.S. Patent

Oct. 18, 2016

Sheet 14 of 18

US 9,471,287 B2

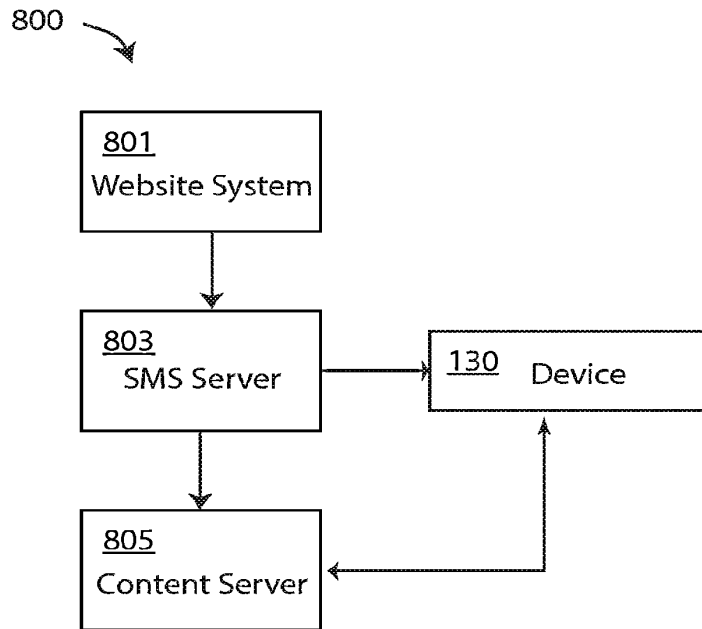


FIG. 8

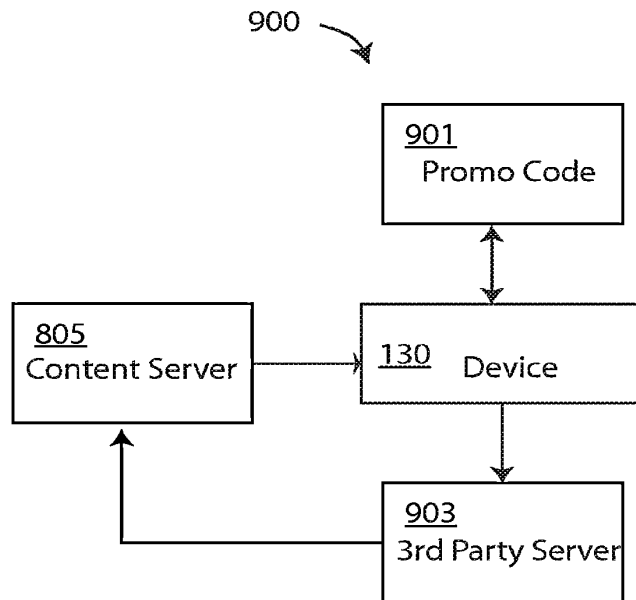


FIG. 9

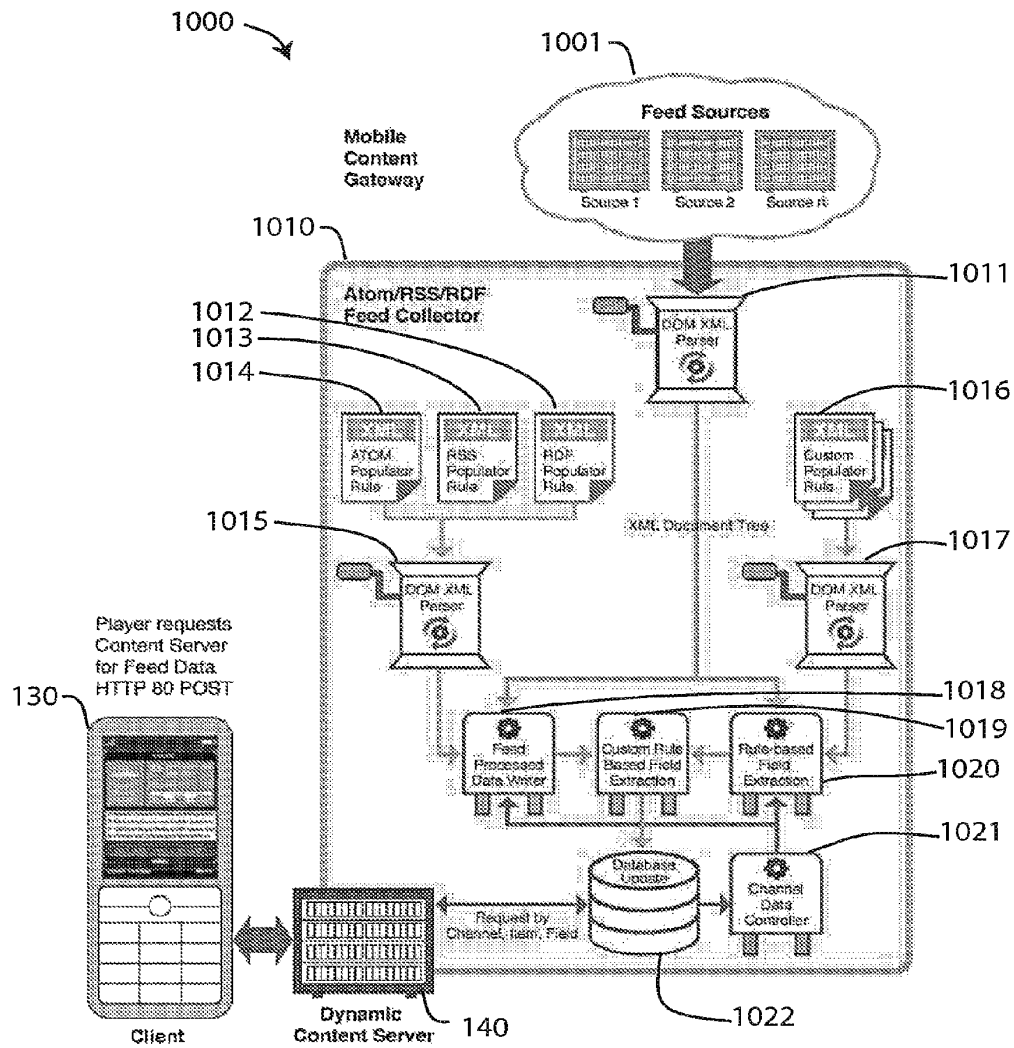
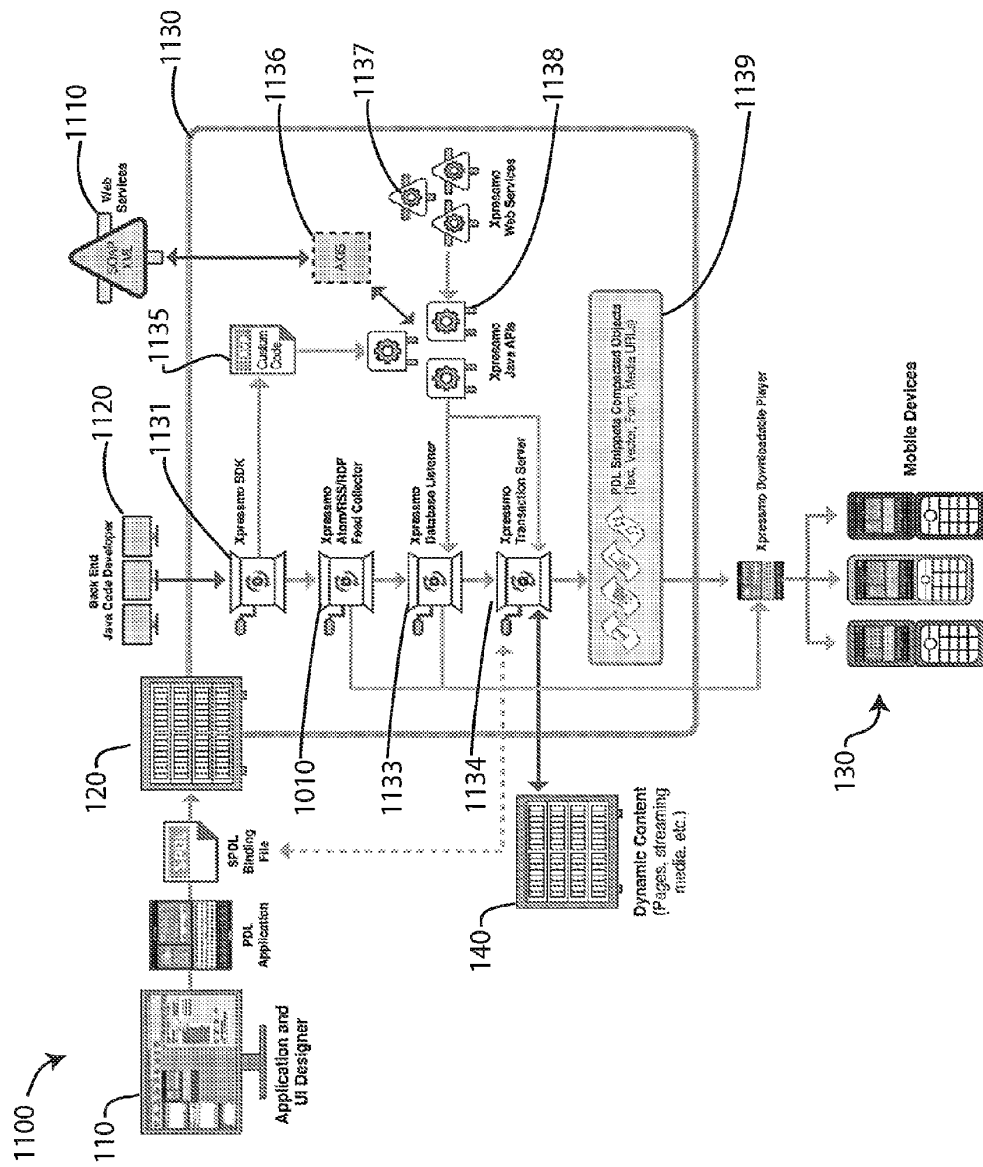


FIG. 10



11

U.S. Patent

Oct. 18, 2016

Sheet 17 of 18

US 9,471,287 B2

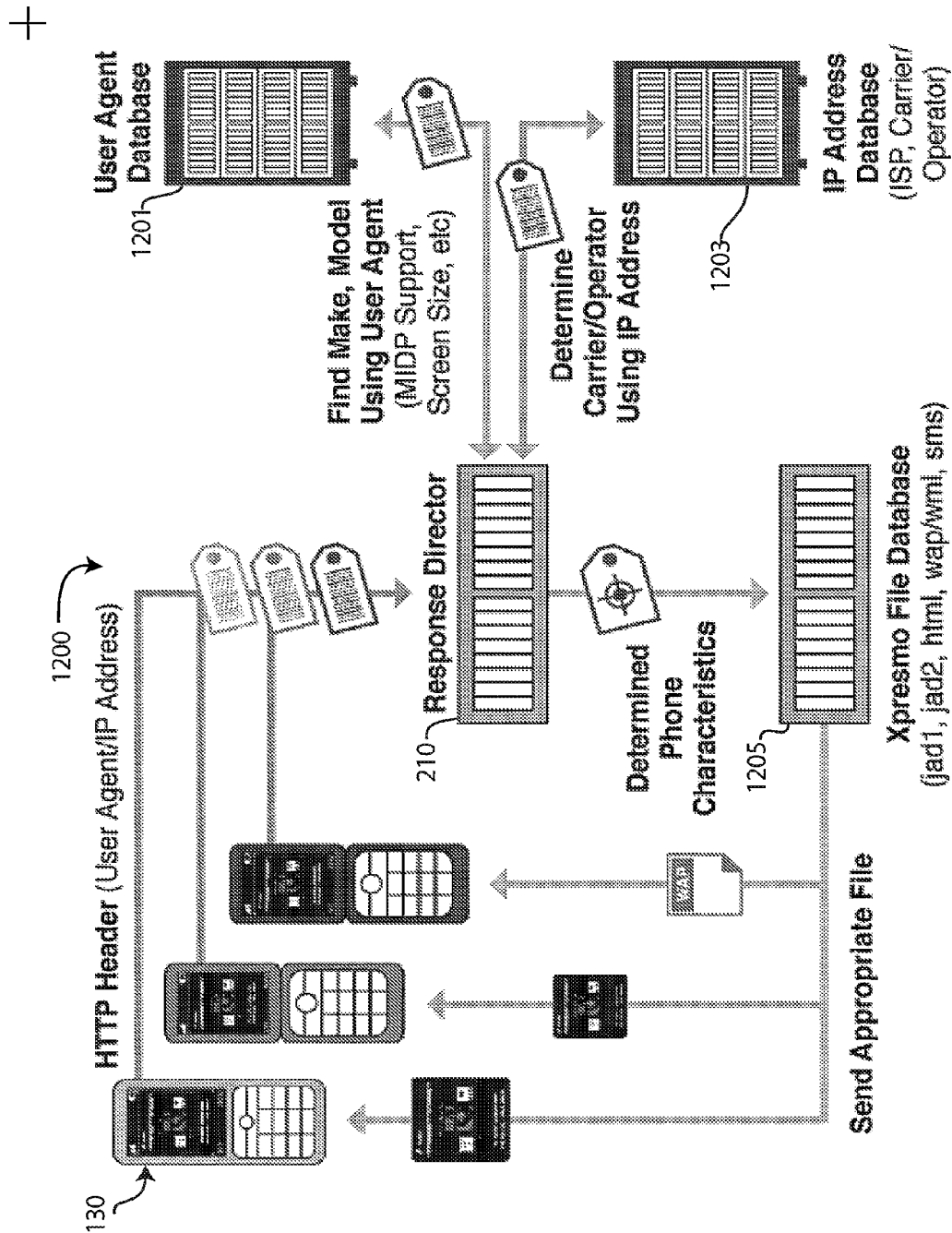


FIG. 12

U.S. Patent

Oct. 18, 2016

Sheet 18 of 18

US 9,471,287 B2

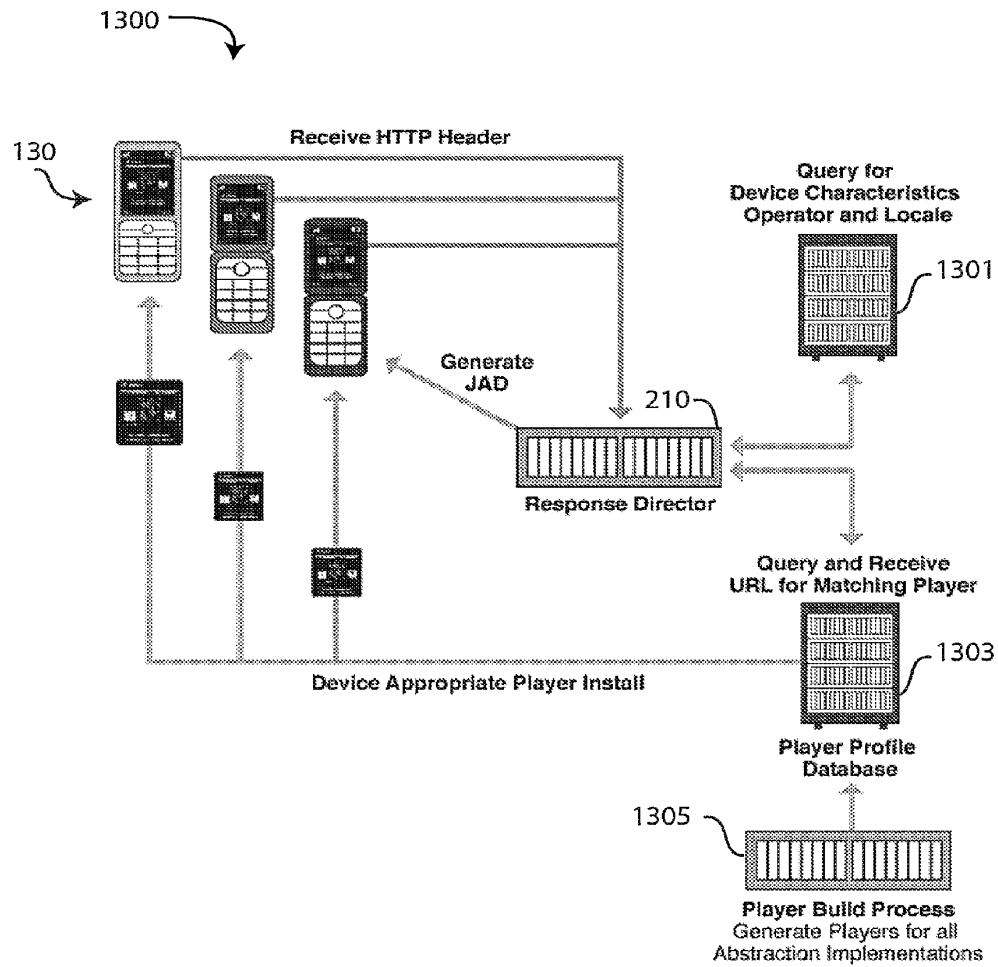


FIG. 13

US 9,471,287 B2

# 1

## SYSTEMS AND METHODS FOR INTEGRATING WIDGETS ON MOBILE DEVICES

### TECHNICAL FIELD

The present invention generally relates to providing software for mobile devices, and more particularly to a method and system for authoring Applications for devices.

### BACKGROUND ART

Internet-connected mobile devices are becoming ever more popular. While these devices provide portability to the Internet, they generally do not have the capabilities of non-mobile devices including computing, input and output capabilities.

In addition, the mobility of the user while using such devices provides challenges and opportunities for the use of the Internet. Further, unlike non-mobile devices, there are a large number of types of devices and they tend to have a shorter lifetime in the marketplace. The programming of the myriad of mobile devices is a time-consuming and expensive proposition, thus limiting the ability of service providers to update the capabilities of mobile devices.

Thus there is a need in the art for a method and apparatus that permits for the efficient programming of mobile devices. Such a method and apparatus should be easy to use and provide output for a variety of devices.

### DISCLOSURE OF INVENTION

In certain embodiments, a system is provided to generate code to provide content on a display of a platform. The system includes a database of web services obtainable over a network and an authoring tool. The authoring tool is configured to define an object for presentation on the display, select a component of a web service included in said database, associate said object with said selected component, and produce code that, when executed on the platform, provides said selected component on the display of the platform.

In certain other embodiments, a method is provided for providing information to platforms on a network. The method includes accepting a first code over the network, where said first code is platform-dependent; providing a second code over the network, where said second code is platform-independent; and executing said first code and said second code on the platform to provide web components obtained over the network.

In certain embodiments, a method for displaying content on a platform utilizing a database of web services obtainable over a network is provided. The method includes: defining an object for presentation on the display; selecting a component of a web service included in said database; associating said object with said selected component; and producing code that, when executed on the platform, provides said selected component on the display of the platform.

In one embodiment, one of the codes is a Player, which is a thin client architecture that operates in a language that manages resources efficiently, is extensible, supports a robust application model, and has no device specific dependencies. In another embodiment, Player P is light weight and extends the operating system and/or virtual machine of the device to: Manage all applications and application upgrades, and resolve device, operating system, VM and language fragmentation.

2

In another embodiment, one of the codes is an Application that is a device independent code that interpreted by the Player.

These features together with the various ancillary provisions and features which will become apparent to those skilled in the art from the following detailed description, are attained by the system and method of the present invention, preferred embodiments thereof being shown with reference to the accompanying drawings, by way of example only, wherein:

### BRIEF DESCRIPTION OF DRAWINGS

FIG. 1A is an illustrative schematic of one embodiment of a system including an authoring platform and a server for providing programming instructions to a device over a network;

FIG. 1B is schematic of an alternative embodiment system for providing programming instructions to device over a network;

FIG. 2A is a schematic of an embodiment of system illustrating the communications between different system components;

FIG. 2B is a schematic of one embodiment of a device illustrating an embodiment of the programming generated by authoring platform;

FIGS. 3A and 3B illustrate one embodiment of a publisher interface as it appears, for example and without limitation, on a screen while executing an authoring tool;

FIG. 3C illustrates an embodiment of the Events Tab'

FIG. 3D illustrates one embodiment of an Animation Tab;

FIG. 3E illustrates one embodiment of Bindings Tab;

FIG. 3F illustrates one embodiment of a pop-up menu for adding web components;

FIG. 4A shows a publisher interface having a layout on a canvas; and FIG. 4B shows a device having the resulting layout on a device screen;

FIG. 5 shows a display of launch strips;

FIG. 6A is a display of a Channel Selection List;

FIG. 6B is a display of a Widget Selection List;

FIG. 6C is a display of a Phone List;

FIG. 7 shows a display of a mash-up;

FIG. 8 is a schematic of an embodiment of a push capable system;

FIG. 9 is a schematic of an alternative embodiment of a push capable system;

FIG. 10 is a schematic of one embodiment of a feed collector;

FIG. 11 is a schematic of an embodiment of a Mobile Content Gateway;

FIG. 12 is a schematic of one embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database; and

FIG. 13 is a schematic of another embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database.

Reference symbols are used in the Figures to indicate certain components, aspects or features shown therein, with reference symbols common to more than one Figure indicating like components, aspects or features shown therein.

### MODE(S) FOR CARRYING OUT THE INVENTION

FIG. 1A is an illustrative schematic of one embodiment of a system **100** including an authoring platform **110** and a server **120** for providing programming instructions to a



US 9,471,287 B2

3

device 130 over a network N. In one embodiment, device 130 is a wireless device, and network N includes wireless communication to the device. Alternatively, system 100 may provide access over network N to other information, data, or content, such as obtainable as a web service over the Internet. In general, a user of authoring platform 110 may produce programming instructions or files that may be transmitted over network N to operate device 130, including instructions or files that are sent to device 130 and/or server 120. The result of the authoring process is also referred to herein, and without limitation, as publishing an Application.

Embodiments include one or more databases that store information related to one or more devices 130 and/or the content provided to the devices. It is understood that such databases may reside on any computer or computer system on network N, and that, in particular, the location is not limited to any particular server, for example.

Device 130 may be, for example and without limitation, a cellular telephone or a portable digital assistant, includes a network interface 131, a memory 133, a processor 135, a screen 137, and an input device 139. Network interface 131 is used by device 130 to communication over a wireless network, such as a cellular telephone network, a WiFi network or a WiMax network, and then to other telephones through a public switched telephone network (PSTN) or to a satellite, or over the Internet. Memory 133 includes programming required to operate device 130 (such as an operating system or virtual machine instructions), and may include portions that store information or programming instructions obtained over network interface 131, or that are input by the user (such as telephone numbers or images from a device camera (not shown)). In one embodiment screen 137 is a touch screen, providing the functions of the screen and input device 139.

Authoring platform 110 includes a computer or computer system having a memory 111, a processor 113, a screen 115, and an input device 117. It is to be understood that memory 111, processor 113, screen 115, and input device 117 are configured such a program stored in the memory may be executed by the processor to accept input from the input device and display information on the screen. Further, the program stored in memory 111 may also instruct authoring platform 110 to provide programming or information, as indicated by the line labeled "A" and to receive information, as indicated by the line labeled "B."

Memory 111 is shown schematically as including a stored program referred to herein, and without limitation, as an authoring tool 112. In one embodiment, authoring tool 112 is a graphical system for designing the layout of features as a display that is to appear on screen 137. One example of authoring tool 112 is the CDER™ publishing platform (Express Mobile, Inc., Novato, Calif.).

In another embodiment, which is not meant to limit the scope of the present invention, device 130 may include an operating system having a platform that can interpret certain routines. Memory 111 may optionally include programming referred to herein, and without limitation, as routines 114 that are executable on device 130.

Routines 114 may include device-specific routines—that is, codes that are specific to the operating system, programming language, or platform of specific devices 130, and may include, but are not limited to, Java, Windows Mobile, Brew, Symbian OS, or Open Handset Alliance (OHA). Several examples and embodiments herein are described with reference to the use of Java. It is to be understood that the invention is not so limited, except as provided in the claims, and that one skilled in the art could provide Players for

4

devices using routines provided on a platform. Thus as an example, routines 114 may include Java API's and an authoring tool System Development Kit (SDK) for specific devices 130.

Server 120 is a computer or computer system that includes a network interface 121, a memory 123, and a processor 125. Is to be understood that network interface 121, memory 123, and processor 125 are configured such that a program stored in the memory may be executed by the processor to: accept input and/or provide output to authoring platform 110; accept input and/or provide output through network interface 121 over network N to network interface 131; or store information from authoring platform 110 or from device 130 for transmission to another device or system at a later time.

In one embodiment, authoring platform 110 permits a user to design desired displays for screen 137 and actions of device 130. In other words, authoring platform 110 is used to program the operation of device 130. In another embodiment, authoring platform 110 allows a user to provide input for the design of one or more device displays and may further allow the user to save the designs as device specific Applications. The Applications may be stored in memory 123 and may then be sent, when requested by device 130 or when the device is otherwise accessible, over network N, through network interface 130 for storage in memory 133.

In an alternative embodiment, analytics information from devices 130 may be returned from device 130, through network N and server 120, back to authoring platform 110, as indicated by line B, for later analysis. Analytics information includes, but is not limited to, user demographics, time of day, and location. The type of analytic content is only limited by which listeners have been activated for which objects and for which pages. Analytic content may include, but is not limited to, player-side page view, player-side forms-based content, player-side user interactions, and player-side object status.

Content server 140 is a computer or computer system that includes a network interface 141, a memory 143, and a processor 145. It is to be understood that network interface 141, memory 143, and processor 145 are configured such that a stored program in the memory may be executed by the processor to accepts requests R from device 130 and provide content C over a network, such as web server content the Internet, to device 130.

FIG. 1B is schematic of an alternative embodiment system 100 for providing programming instructions to device 130 over a network N that is generally similar to the system of FIG. 1A. The embodiment of FIG. 1B illustrates that system 100 may include multiple servers 120 and/or multiple devices 130.

In the embodiment of FIG. 1B, system 100 is shown as including two or more servers 120, shown illustratively and without limitation as servers 120a and 120b. Thus some of the programming or information between authoring platform 110 and one or more devices 130 may be stored, routed, updated, or controlled by more than one server 120. In particular, the systems and methods described herein may be executed on one or more server 120.

Also shown in FIG. 1B are a plurality of devices 130, shown illustratively and without limitation as device 130-1, 130-1, . . . 130-N. System 100 may thus direct communication between individual server(s) 120 and specific device(s) 130.

As described subsequently, individual devices 130 may be provided with program instructions which may be stored in each device's memory 133 and where the instructions are

US 9,471,287 B2

5

executed by each device's processor 135. Thus, for example, server(s) 120 may provide device(s) 130 with programming in response to the input of the uses of the individual devices. Further, different devices 130 may be operable using different sets of instructions, that is having one of a variety of different "device platforms." Differing device platforms may result, for example and without limitation, to different operating systems, different versions of an operating system, or different versions of virtual machines on the same operating system. In some embodiments, devices 130 are provided with some programming from authoring system 100 that is particular to the device.

In one embodiment, system 100 provides permits a user of authoring platform 110 to provide instructions to each of the plurality of devices 130 in the form of a device- or device-platform specific instructions for processor 135 of the device, referred to herein and without limitation as a "Player," and a device-independent program, referred to herein and without limitation as an "Application." Thus, for example, authoring platform 110 may be used to generate programming for a plurality of devices 130 having one of several different device platforms. The programming is parsed into instructions used by different device platforms and instructions that are independent of device platform. Thus in one embodiment, device 130 utilizes a Player and an Application to execute programming from authoring platform 110. A device having the correct Player is then able to interpret and be programmed according to the Application.

In one alternative embodiment, the Player is executed the first time by device 130 ("activated") through an Application directory. In another alternative embodiment, the Player is activated by a web browser or other software on device 130. In yet another alternative embodiment, Player is activated through a signal to device 130 by a special telephone numbers, such as a short code.

When the Application and the Player are provided to memory 133, the functioning of device 130 may occur in accordance with the desired programming. Thus in one embodiment, the Application and Player includes programming instructions which may be stored in memory 133 and which, when executed by processor 135, generate the designed displays on screen 137. The Application and Player may also include programming instructions which may be stored in memory 133 and which provide instructions to processor 135 to accept input from input device 139.

Authoring tool 112 may, for example, produce and store within memory 111 a plurality of Players (for different devices 130) and a plurality of Applications for displaying pages on all devices. The Players and Applications are then stored on one or more servers 120 and then provided to individual devices 130. In general, Applications are provided to device 130 for each page of display or a some number of pages. A Player need be provided once or updated as necessary, and thus may be used to display a large number of Applications. This is advantageous for the authoring process, since all of the device-dependent programming is provided to a device only once (or possibly for some small number of upgrades), permitting a smaller Application, which is the same for each device 130.

Thus, for example and without limitation, in one embodiment, the Player transforms device-independent instructions of the Application into device-specific instructions that are executable by device 130. Thus, by way of example and without limitation, the Application may include Java programming for generating a display on screen 137, and the Player may interpret the Java and instruct processor 135 to produce the display according to the Application for execu-

6

tion on a specific device 130 according to the device platform. The Application may in general include, without limitation, instructions for generating a display on screen 137, instructions for accepting input from input device 139, instructions for interacting with a user of device 130, and/or instructions for otherwise operating the device, such as to place a telephone call.

The Application is preferably code in a device-independent format, referred to herein and without limitation as a Portable Description Language (PDL). The device's Player interprets or executes the Application to generate one or more "pages" ("Applications Pages") on a display as defined by the PDL. The Player may include code that is device-specific—that it, each device is provided with a Player that is used in the interpretation and execution of Applications. Authoring tool 112 may thus be used to design one or more device-independent Applications and may also include information on one or more different devices 130 that can be used to generate a Player that specific devices may use to generate displays from the Application.

In one embodiment, system 100 provides Players and Applications to one server 120, as in FIG. 1A. In another embodiment, system 100 provides Players to a first server 120a and Applications to a second server 120b, as in FIG. 1B.

In one embodiment, authoring tool 112 may be used to program a plurality of different devices 130, and routines 114 may include device-specific routines. In another embodiment, the Player is of the type that is commonly referred to as a "thin client"—that is, software for running on the device as a client in client-server architecture with a device network which depends primarily on a central server for processing activities, and mainly focuses on conveying input and output between the user and the server.

In one embodiment, authoring platform 110 allows user to arrange objects for display on screen. A graphical user interface ("GUI," or "UI") is particularly well suited to arranging objects, but is not necessary. The objects may correspond to one or more of an input object, an output object, an action object, or may be a decorative display, such as a logo, or background color or pattern, such as a solid or gradient fill. In another embodiment, authoring platform 110 also permits a user to assign actions to one or more of an input object, an output object, or an action object. In yet another embodiment, authoring platform 110 also permits a user to bind one or more of an input object, an output object, or an action object with web services or web components, or permits a user to provide instructions to processor 135 to store or modify information in memory 133, to navigate to another display or service, or to perform other actions, such as dialing a telephone number.

In certain embodiments, the applicant model used in developing and providing Applications is a PDL. The PDL can be conceptually viewed as a device, operating system and virtual machine agnostic representation of Java serialized objects. In certain embodiments, the PDL is the common language for authoring tool 112, the Application, and Player. Thus while either designing the Application with the authoring tool 112, or programming with the SDK, the internal representation of the programming logic is in Java. In one embodiment the SDK is used within a multi-language software development platform comprising an IDE and a plug-in system to extend it, such as the Eclipse Integrated Development Environment (see, for example, <http://www.eclipse.org/>). At publish time the Java code is translated into

US 9,471,287 B2

7

a PDL. This translation may also occur in real-time during the execution of any Web Services or backend business logic that interacts with the user.

One embodiment for compacting data that may be used is described in co-pending U.S. Pat. No. 6,546,397 to Rempell ("Rempell"), the contents of which are incorporated herein by reference. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

The use of a PDL, as described in Rempell, permits for efficient code and data compaction. Code, as well as vector, integer and Boolean data may be compacted and then compressed resulting in a size reduction of 40 to 80 times that of the original Java serialized objects. This is important not only for performance over the network but for utilizing the virtual memory manager of the Player more efficiently. As an example, the reassembled primitives of the Java objects may first undergo logical compression, followed by LZ encoding.

The use of a PDL also provides virtual machine and operating system independence. Since the reassembled primitives of the Application no longer have any dependencies from the original programming language (Java) that they were defined in. The PDL architecture takes full advantage of this by abstracting all the virtual machine and/or operating system interfaces from the code that processes the PDL.

In one embodiment, the PDL is defined by the means of nested arrays of primitives. Accordingly, the use of a PDL provides extensibility and compatibility, with a minimal amount of constraints in extending the Player seamlessly as market demands and device capabilities continue to grow. Compatibility with other languages is inherent based on the various Player abstraction implementations, which may be, for example and without limitation, Java CDC, J2SE or MIDP2 implementations.

In one embodiment, the architecture of Player P includes an abstraction interface that separates all device, operating system and virtual machine dependencies from the Player's Application model business logic (that is, the logic of the server-side facilities) that extend the Application on the Player so that it is efficiently integrated into a comprehensive client/server Application. The use of an abstraction interface permits the more efficient porting to other operating systems and virtual machines and adding of extensions to the Application model so that a PDL can be implemented once and then seamlessly propagated across all platform implementations. The Application model includes all the currently supported UI objects and their attributes and well as all of the various events that are supported in the default Player. Further, less robust platforms can be augmented by extending higher end capabilities inside that platform's abstraction interface implementation.

In one embodiment, authoring platform 110 provides one or more pages, which may be provided in one Application, or a plurality of Applications, which are stored in memory 123 and subsequently provided to memory 133. In certain embodiments, the Application includes instructions R to request content or web services C from content server 140. Thus, for example and without limitation, the request is for information over the network via a web service, and the request R is responded to with the appropriate information for display on device 130. Thus, for example, a user may request a news report. The Application may include the layout of the display, including a space for the news, which is downloaded from content server 140 for inclusion on the

8

display. Other information that may be provided by content server 140 may include, but is not limited to, pages, Applications, multimedia, and audio.

FIG. 2A is a schematic of a system 200 of an embodiment of system 100 illustrating the communications between different system components. System includes a response director 210, a web component registry 220, and a web service 230. System 200 further includes authoring platform 110, server 120, device 130 and content server 140 are which are generally similar to those of the embodiments of FIGS. 1A and 1B, except as explicitly noted.

Response director 210 is a computer or computer system that may be generally similar to server 120 including the ability to communicate with authoring platform 110 and one or more devices 130. In particular, authoring platform 110 generates one or more Players (each usable by certain devices 130) which are provided to response director 210. Devices 130 may be operated to provide response director 210 with a request for a Player and to receive and install the Player. In one embodiment, device 130 provides response director 210 with device-specific information including but not limited to make, model, and/or software version of the device. Response director 210 then determines the appropriate Player for the device, and provides the device with the Player over the network.

Web service 230 is a plurality of services obtainable over the Internet. Each web service is identified and/or defined as an entry in web component registry 230, which is a database, XML file, or PDL that exists on a computer that may be a server previously described or another server 120. Web component registry 230 is provided through server 120 to authoring platform 110 so that a user of the authoring platform may bind web services 230 to elements to be displayed on device 130, as described subsequently.

In one embodiment, authoring platform 110 is used in conjunction with a display that provides a WYSIWYG environment in which a user of the authoring platform can produce an Application and Player that produces the same display and the desired programming on device 130. Thus, for example, authoring tool 112 provides a display on screen 115 that corresponds to the finished page that will be displayed on screen 137 when an Application is intercepted, via a Player, on processor 135 of device 130.

Authoring platform 110 further permits a user of the authoring platform to associate objects, such as objects for presenting on screen 137, with components of one or more web services 230 that are registered in web component registry 220. In one embodiment, information is provided in an XML file to web component registry 220 for each registered components of each web service 230. Web component registry 220 may contain consumer inputs related to each web service 230, environmental data such as PIM, time or location values, persistent variable data, outputs related to the web service, and/or optional hinting for improving the user's productivity.

A user of authoring platform 110 of system 200 may define associations with web services as WebComponent Bindings. In one embodiment, authoring platform 110 allows a user to associate certain objects for display that provide input or output to components of web service 230. The associated bindings are saved as a PDL in server 120.

In one embodiment, an XML web component registry 220 for each registered web service 230 is loaded into authoring platform 110. The user of system 200 can then assign components of any web service 230 to an Application without any need to write code. In one embodiment, a component of web service 230 is selected from authoring

US 9,471,287 B2

9

platform 110 which presents the user with WYSIWYG dialog boxes that enable the binding of all the inputs and outputs of component of web service 230 to a GUI component of the Application as will be displayed on screen 137. In addition, multiple components of one or more web service 230 can be assigned to any Object or Event in order to facilitate mashups. These Object and/or Event bindings, for each instance of a component of any web service 230, are stored in the PDL. The content server 140 handles all communication between device 130 and the web service 230 and can be automatically deployed as a web application archive to any content server.

Device 130, upon detecting an event in which a component of a web service 230 has been defined, assembles and sends all related inputs to content server 240, which proxies the request to web service 230 and returns the requested information to device 130. The Player on device 130 then takes the outputs of web service 230 and binds the data to the UI components in the Application, as displayed on screen 137.

In one embodiment, the mechanism for binding the outputs of the web service to the UI components is through symbolic references that matches each output to the symbolic name of the UI component. The outputs, in one embodiment, may include meta-data which could become part of the inputs for subsequent interactions with the web service.

For example, if a user of authoring platform 110 wants to present an ATOM feed on device 130, they would search through a list of UI Components available in the authoring platform, select the feed they want to use, and bind the output of the feed summary to a textbox. The bindings would be saved into the PDL on server 120 and processed by device 130 at runtime. If the ATOM feed does not exist a new one can be added to the web component registry that contains all the configuration data required, such as the actual feed URL, the web component manager URL, and what output fields are available for binding.

In another embodiment, components of web services 230 are available either to the user of authoring platform 110 or otherwise accessible through the SDK and Java APIs of routines 114. System 200 permits an expanding set of components of web services 230 including, but not limited to: server pages from content server 120; third-party web services including, but not limited to: searching (such through Google or Yahoo), maps (such as through MapQuest and Yahoo), storefronts (such as through ThumbPlay), SMS share (such as through clickatel), stock quotes, social networking (such as through FaceBook), stock quotes, weather (such as through Accuweather) and/or movie trailers. Other components include web services for communication and sharing through chats and forums and rich messaging alerts, where message alerts are set-up that in turn could have components of Web Services 230 defined within them, including the capture of consumer generated and Web Service supplied rich media and textual content.

System 200 also permits dynamic binding of real-time content, where the inputs and outputs of XML web services are bound to GUI components provided on screen 137. Thus, for example, a user of authoring platform 110 may bind attributes of UI Objects to a particular data base field on a Server. When running the Application, the current value in the referenced data base will be immediately applied. During the Application session, any other real time changes to these values in the referenced data base will again be immediately displayed.

10

As an example of dynamic binding of real-time content, an RSS feeds and other forms of dynamic content may be inserted into mobile Applications, such as device 130, using system 200. Authoring platform 110 may include a "RSS display" list which permits a user to select RSS channels and feeds from an extensible list of available dynamic content. Meta data, such as titles, abstracts and Images can be revealed immediately by the user as they traverse this RSS display list, bringing the PC experience completely and conveniently to mobile devices 130. In addition, Authoring platform 110 may include a dialog box that dynamically links objects to data and feeds determined by RSS and chat databases. Any relevant attribute for a page view and/or object can be dynamically bound to a value in a server-side database. This includes elements within complex objects such as: any icon or text element within a graphical list; any icon within a launch strip; any feature within any geographical view of a GIS service object; and/or any virtual room within a virtual tour.

As an example of third-party web services 230 that may be provided using system 200, a user of authoring platform 110 can place, for example, Yahoo maps into device 130 by binding the required component of the Yahoo Maps Web Service, such as Yahoo Map's Inputs and/or Outputs to appropriate Objects of authoring platform 110. System 200 also provides binding to web services for text, image and video searching by binding to components of those web services.

In one embodiment, an Application for displaying on device 130 includes one or more Applications Pages, each referred to herein as an "XSP," that provides functionality that extends beyond traditional web browsers. The XSP is defined as a PDL, in a similar manner as any Application, although it defines a single page view, and is downloaded to the Player dynamically as required by the PDL definition of the Application. Thus, for example, while JSPs and ASPs, are restricted to the functionality supported by the web browser, the functionality of XSPs can be extended through authoring platform 110 having access to platform dependent routines 114, such as Java APIs. Combined with dynamic binding functionality, an XSP, a page can be saved as a page object in an author's "pages" library, and then can be dynamically populated with real-time content simultaneously as the page is downloaded to a given handset Player based on a newly expanded API. XSP Server Pages can also be produced programmatically, but in most cases authoring platform 110 will be a much more efficient way to generate and maintain libraries of dynamically changing XSPs.

With XSPs, Applications Pages that have dynamic content associated with them can be sent directly to device 130, much like how a web browser downloads an HTML page through an external reference. Without XSPs, content authors would have to define each page in the Application. With XSPs, no pages need to be defined. Thus, for example, in a World Cup Application, one page could represent real-time scores that change continuously on demand. With polling (for example, a prompt to the users asking who they predict will win a game), a back-end database would tabulate the information and then send the results dynamically to the handsets. With a bar chart, the Application would use dynamic PDL with scaling on the fly. For example, the server would recalibrate the bar chart for every ten numbers.

Other combinations of components of web services 230 include, but are not limited to, simultaneous video chat sessions, inside an integrated page view, with a video or television station; multiple simultaneous chat sessions, each

US 9,471,287 B2

11

with a designated individual and/or group, with each of the chat threads visible inside an integrated page view.

Another extension of an XSP is a widget object. Widgets can be developed from numerous sources including, but not limited to, authoring platform 110, a Consumer Publishing Tool, and an XML to Widget Conversion Tool where the SDK Widget Libraries are automatically populated and managed, or Widget Selection Lists that are available and can be populated with author defined icons.

Applications, Players, and Processing in a Device

FIG. 2B is a schematic of one embodiment of a device 130 illustrating an embodiment of the programming generated by authoring platform 110. Memory 133 may include several different logical portions, such as a heap 133a, a record store 133b and a filesystem (not shown).

As shown in FIG. 2B, heap 133a and record store 133b include programming and/or content. In general, heap 133a is readily accessible by processor 135 and includes, but is not limited to portions that include the following programming: a portion 133a1 for virtual machine compliant objects representing a single Page View for screen 137; a portion 133a2 for a Player; a portion 133a3 for a virtual machine; and a portion 133a4 for an operating system.

Record store 133b (or alternatively the filesystem) includes, but is not limited to, portions 133b1 for Applications and non-streaming content, which may include portions 133a2 for images, portions 133a4 for audio, and/or portions 133a5 for video, and portions 133b2 for non-Application PDLs, such as a Master Page PDL for presenting repeating objects, and Alerts, which are overlayed on the current page view. Other content, such as streaming content may be provided from network interface 131 directly to the Media Codec of device 130 with instructions from Player on how to present the audio or video.

In one embodiment, the Player includes a Threading Model and a Virtual Memory Manager. The Threading Model first manages a queue of actions that can be populated based on Input/Output events, Server-side events, time-based events, or events initiated by user interactions. The Threading Model further manages the simultaneous execution of actions occurring at the same time. The Virtual Memory Manager includes a Logical Virtual Page controller that provides instructions from the record store to the heap, one page at a time. Specifically, the Virtual Memory Manager controls the transfer of one of the Application Pages and its virtual machine compliant objects into portion 133a1 as instructions readable by the Player or Virtual Machine. When the Player determines that a new set of instructions is required, the information (such as one Application Page is retrieved from the Record store, converted into virtual machine compliant objects (by processor 135 and according to operation by the Player, Virtual Machine, etc), and stored in heap 133a. Alternatively, the Player may augment virtual machine compliant objects with its own libraries for managing user interactions, events, memory, etc.

The connection of portions 133a1, 133a2, 133a3, 133a4, record store 133b and processor 135 are illustrative of the logical connection between the different types of programming stored in Heap 133a and record store 133b, that is, how data is processed by processor 135.

The Player determines which of the plurality of Application Pages in portion 133b1 is required next. This may be determined by input actions from the Input Device 139, or from instructions from the current Application Page. The Player instructs processor 135 to extract the PDF from that Applications Page and store it in portion 133a1. The Player then interprets the Application Page extracted from PDL

12

which in turn defines all of the virtual machine compliant Objects, some of which could have attributes that refer to images, audio, and/or video stored in portions 133a3, 133a4, 133a5, respectively.

The Virtual Machine in portion 133a3 processes the Player output, the Operating System in portion 133a3 processes the Virtual Machine output which results in machine code that is processed by the Operating System in portion 133a4.

In another embodiment, the Player is a native program that interacts directly with the operating system.

Embodiments of a Publishing Environment

In one embodiment, authoring platform 110 includes a full-featured authoring tool 112 that provides a what-you-see-is-what-you-get (WYSIWYG) full featured editor. Thus, for example, authoring tool 112 permits a user to design an Application by placing objects on canvas 305 and optionally assigning actions to the objects and save the Application. System 100 then provides the Application and Player to a device 130. The Application as it runs on device 130 has the same look and operation as designed on authoring platform 110. In certain embodiments, authoring platform 110 is, for example and without limitation, a PC-compatible or a Macintosh computer.

Authoring platform 110 produces an Application having one or more Applications Pages, which are similar to web pages. That is, each Applications Page, when executed on device 130 may, according to its contents, modify what is displayed on screen 137 or cause programming on the device to change in a manner similar to how web pages are displayed and navigated through on a website.

In one embodiment, authoring tool 112 allows a user to place one or more objects on canvas 305 and associate the objects with an Applications Pages. Authoring platform 110 maintains a database of object data in memory 111, including but not limited to type of object, location on which page, and object attributes. The user may add settings, events, animations or binding to the object, from authoring tool 112, which are also maintained in memory 111. Authoring tool 112 also allows a user to define more than one Applications Page.

In another embodiment, authoring tool 112, provides Java programming functions of the Java API for specific devices 130 as pull-down menus, dialog boxes, or buttons. This permits a user of authoring platform 110 to position objects that, after being provided as an Application to device 130, activate such Java functions on the device.

In certain embodiments, authoring platform 110, as part of system 100, permits designers to include features of advanced web and web services Applications for access by users of device 130. Some of the features of advanced web and web services include, but are not limited to: slide shows, images, video, audio, animated transitions, multiple chats, and mouse interaction; full 2-D vector graphics; GIS (advanced LBS), including multiple raster and vector layers, feature sensitive interactions, location awareness, streaming and embedded audio/video, virtual tours, image processing and enhancement, and widgets. In other embodiments the features are provided for selection in authoring platform 110 through interactive object libraries.

In certain embodiments, authoring platform 110, as part of system 100, allows the inclusion of child objects which may eventually be activated on device 130 by the user of the device or by time. The uses of the child objects on device 130 include, but are not limited to: mouse over (object selection), hover and fire events and launching of object-specific, rich-media experiences.

US 9,471,287 B2

13

In certain other embodiments, authoring platform 110, as part of system 100, provides advanced interactive event models on device 130, including but not limited to: user-, time- and/or location-initiated events, which allow content developers to base interactivity on specific user interactions and/or instances in time and space; timelines, which are critical for timing of multiple events and for animations when entering, on, or exiting pages of the Application; waypoints, which act similar to key frames, to allow smooth movement of objects within pages of the Application. Waypoints define positions on a page object's animation trajectory. When an object reaches a specific waypoint other object timelines can be initiated, creating location-sensitive multiple object interaction, and/or audio can be defined to play until the object reaches the next waypoint.

Authoring platform 110 may also define a Master Page, which acts as a template for an Applications Page, and may also define Alert Pages, which provide user alerts to a user of device 130.

In certain embodiments, authoring platform 110, as part of system 100, provides full style inheritance on device 130. Thus, for example and without limitation, both master page inheritance (for structural layout inheritance and repeating objects) and object styles (for both look and feel attribute inheritance) are supported. After a style has been defined for an object, the object will inherit the style. Style attributes include both the look and the feel of an object, including mouse interaction, animations, and timelines. Each page may include objects that may be a parent object or a child object. A child object is one that was created by first selecting a parent object, and then creating a child object. Child objects are always part of the same drawing layer as its parent object, but are drawn first, and are not directly selectable when running the Application. A parent object is any object that is not a child object, and can be selected when running the Application.

As an example, the user of authoring tool 112 may create various text objects on canvas 305 using a style that sets the font to red, the fonts of these objects will be red. Suppose user of authoring tool 112 changes the font color of a specific button to green. If later, the user of authoring tool 112 changes the style to blue; all other text objects that were created with that style will become blue except for the button that had been specifically set to green.

In certain other embodiments, authoring platform 110 provides page view, style, object, widget and Application template libraries. Authoring platform 110 may provide templates in private libraries (available to certain users of the authoring platform) and public libraries (available to all users of the authoring platform). Templates may be used to within authoring platform 110 to define the look and feel of the entire Application, specific pages, or specific slide shows and virtual tours as seen on device 130.

FIGS. 3A and 3B illustrate one embodiment of a publisher interface 300 as it appears, for example and without limitation, on screen 115 while executing authoring tool 112. In one embodiment, publisher interface 300 includes a Menu bar 301, a Tool bar 303, a Canvas 305, a Layer Inspector 307 having subcomponents of a page/object panel 307a, an object style panel 307b, and a page alert panel 307c, and a Resource Inspector 309.

In general, publisher interface 300 permits a user of authoring platform 110 to place objects on canvas 305 and then associate properties and/or actions to the object, which are stored in the Application. As described subsequently, publisher interface 300 permits a user to program a graphical interface for the screen 137 of device 130 on screen 115 of

14

authoring platform 110, save an Application having the programming instructions, and save a Player for the device. The intended programming is carried out on device 130 when the device, having the appropriate device platform Player, receives and executes the device-independent Application.

Thus, for example, authoring tool 112 maintains, in memory 111, a list of every type of object and any properties, actions, events, or bindings that may be assigned to that object. As objects are selected for an Application, authoring tool 112 further maintains, in memory 111, a listing of the objects. As the user selects objects, publisher interface 300 provides the user with a choice of further defining properties, actions, events, or bindings that may be assigned to each particular object, and continues to store the information in memory 111.

In one embodiment, publisher interface 300 is a graphical interface that permits the placement and association of objects in a manner typical of, for example, vector graphics editing programs (such as Adobe Illustrator). Objects located on canvas 305 placed and manipulated by the various commands within publisher interface 300 or inputs such as an input device 117 which may be a keyboard or mouse. As described herein, the contents of canvas 305 may be saved as an Application that, through system 100, provide the same or a similar placement of objects on screen 137 and have actions defined within publisher interface 300. Objects placed on canvas 305 are intended for interaction with user of device 130 and are referred to herein, without limitation, as objects or UI (user interface) objects. In addition, the user of interface 300 may assign or associate actions or web bindings to UI objects placed on canvas 305 with result in the programming device 130 that cause it to respond accordingly.

Objects include, but are not limited to input UI objects, response UI objects. Input UI objects include but are not limited to: text fields (including but not limited to alpha, numeric, phone number, or SMS number); text areas; choice objects (including but not limited to returning the selected visible string or returning a numeric hidden attribute); single item selection lists (including but not limited to returning the selected visible string or returning a numeric hidden attribute); multi item selection lists (including but not limited to returning all selected items (visible text string or hidden attribute) or cluster item selection lists (returning the hidden attributes for all items).

Other input UI objects include but are not limited to: check boxes; slide show (including but not limited to returning a numeric hidden attribute, returning a string hidden attribute, or returning the hidden attributes for all slides); and submit function (which can be assigned to any object including submit buttons, vectors, etc.).

Response UI Objects may include, but are not limited to: single line text objects, which include: a text Field (including but not limited to a URL, audio URL, or purchase URL), a text button, a submit button, or a clear button. Another response UI objects include: a multiple line text object, which may include a text area or a paragraph; a check box; an image; a video; a slide show (with either video or image slides, or both); choice objects; list objects; or control lists, which control all the subordinate output UI objects for that web component. Control list objects include, but are not limited to: list type or a choice type, each of which may include a search response list or RSS display list.

As a further example of objects that may be used with authoring tool 112, Table I lists Data Types, Preferred Input, Input Candidates, Preferred Output and Output Candidates for one embodiment of an authoring tool.

US 9,471,287 B2

15

16

TABLE I

One embodiment of supported objects				
Data Types	Preferred Input	Input Candidates	Preferred Output	Output Candidates
boolean	Check Box	Check Box	Check Box	Check Box
Int	Text Field (integer)	Text Field (integer)	Text Field (integer)	Text Field (integer)
		Text Field (Phone #)		Text Field (Phone #)
		Text Field (SMS #)		Text Field (SMS #)
		Choice		Choice
		List (single select)		List (single select)
String	Text Field (Alpha)	Any	Text Field (Alpha)	Any
multilineString	Text Area	Text Area	Text Area	Text Area
ImageURL	N/A	N/A	Image	Paragraph
VideoURL	N/A	N/A	Image	Image
			Video	Slide Show
List	Single Item List	Single Item List	Single Item List	Video
		Multi-Select List		Slide Show
		Complex List		Any List Type
		Choice		Any Choice Type
		Slide Show		(see Complex List Specification)
ComplexList	Complex List	Single Item List	Single Item List	Any List Type
		Multi-Select List		(see Complex List Specification)
Slideshow	Slide Show	Complex List	Slide Show	Slide Show
SearchResponseList	N/A	Slide Show	Search Response List	Search Response List
		N/A		Control List
				Complex List
				Choice
RSSList	N/A	N/A	RSS Display List	RSS Display List
				Control List
				Complex List
				Choice
SingleSelectionList	Choice	Choice	Choice	Choice
		Complex List		Complex List
MultiSelectionList	Multi-Selection List	Multi-Selection List	Multi-Selection List	Multi-Selection List
ServiceActivation	Submit Button	Any	N/A	N/A
ChannelImageURL	N/A	N/A	Image	Image
				Video
				Slide Show
ChannelDescription	N/A	N/A	Text Area	Text Area
				Paragraph
				Text Field
				Text Button
				List
				Choice
ChannelTitle	N/A	N/A	Text Field	Text Field
				Text Button
				Paragraph
				Text Area
				List
				Choice
URL			Text Field	Text Field
			(URL request)	(URL request)
Audio URL			Text Field	Text Field
			(Audio URL request)	(Audio URL request)
Purchase URL			Text Field	Text Field
			(Purchase URL request)	(Purchase URL request)
Image Data			Image	Image
				Slide Show
Image List Data			Slide Show	Slide Show
				Image
Persistent Variable	N/A	N/A	N/A	N/A
Pipeline Multiple Select	Multi-select List	Multi-select List	N/A	N/A
		Complex List		
		Slide Show		
Phone Number	Text Field (numeric type)	Text Field	Text Field (numeric type)	Text Field
		Text Button		Text Button
Hidden Attribute	Complex List	Complex List	Complex List	Complex List
		Slide Show		Slide Show
Collection List	N/A	N/A	Slide Show	Complex List
				Slide Show

US 9,471,287 B2

17

In general, publisher interface **300** permits a user to define an Application as one or more Applications Pages, select UI objects from Menu bar **301** or Tool bar **303** and arrange them on an Applications Page by placing the objects canvas **305**. An Application Page is a page that is available to be visited through any navigation event. Application Pages inherit all the attributes of the Master Page, unless that attribute is specifically changed during an editing session.

Authoring platform **110** also stores information for each UI object on each Application Page of an Application. Layer Inspector **307** provides lists of Applications Pages, UI objects on each Applications Page, and Styles, including templates. Objects may be selected from canvas **305** or Layer Inspector **307** causing Resource Inspector **309** to provide lists of various UI objects attributes which may be selected from within the Resource Inspector. Publisher interface **300** also permits a user to save their work as an Application for layer transfer and operation of device **130**. Publisher interface **300** thus provides an integrated platform for designing the look and operation of device **130**.

The information stored for each UI object depends, in part, on actions which occur as the result of a user of device **130** selecting the UI object from the device. UI objects include, but are not limited to: navigational objects, such as widget or channel launch strips or selection lists; message objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields or a pop-up alert; text fields or areas; check boxes; pull down menus; selection lists and buttons; pictures; slide shows; video or LBS maps; shapes or text defined by a variety of tools; a search response; or an RSS display.

In certain embodiments, publisher interface **300** permits a user to assign action to UI objects, including but not limited to, programming of the device **130** or a request for information over network N. In one embodiment, for example and without limitation, publisher interface **300** has a selection to bind a UI object to a web service—that is, associate the UI object or a manipulation or selection of UI object with web services. Publisher interface **300** may also include many drawing and text input functions for generating displays that may be, in some ways, similar to drawing and/or word processing programs, as well as toolbars and for zooming and scrolling of a workspace.

Each UI object has some form, color, and display location associate with it. Further, for example and without limitation, UI objects may have navigational actions (such as return to home page), communications actions (such as to call the number in a phone number field), or web services (such as to provide and/or retrieve certain information from a web service). Each of these actions requires authoring platform **110** to store the appropriate information for each action. In addition, UI objects may have associated patent or child objects, default settings, attributes (such as being a password or a phone number), whether a field is editable, animation of the object, all of which may be stored by authoring platform **110**, as appropriate.

Menu bar **301** provides access features of publisher interface **300** through a series of pull-down menus that may include, but are not limited to, the following pull-down menus: a File menu **301a**, an Edit menu **301b**, a View menu **301c**, a Project menu **301d**, an Objects menu **301e**, an Events menu **301f**, a Pages menu **301g**, a Styles menu **301h**, and a Help menu **301i**.

File menu **301a** provides access to files on authoring platform **110** and may include, for example and without limitation, selections to open a new Application or master page, open a saved Application, Application template, or

18

style template, import a page, alert, or widget, open library objects including but not limited to an image, video, slide show, vector or list, and copying an Application to a user or to Server **120**.

Edit menu **301b** may include, but is not limited to, selections for select, cut, copy, paste, and edit functions.

View menu **301c** may include, but is not limited to, selections for zooming in and out, previewing, canvas **305** grid display, and various palette display selections.

Project menu **301d** may include, but is not limited to, selections related to the Application and Player, such as selections that require a log in, generate a universal Player, generate server pages, activate server APIs and extend Player APIs. A Universal Player will include all the code libraries for the Player, including those that are not referenced by the current Application. Server APIs and Player APIs logically extend the Player with Server-side or device-side Application specific logic.

Objects menu **301e** includes selections for placing various objects on canvas **305** including, but not limited to: navigation UI objects, including but not limited to widget or channel launch strips or selection lists; message-related UI objects, including but not limited to multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert; shapes, which provides for drawing tools; forms-related objects, including but not limited to text fields; scrolling text box, check box, drop-down menu, list menu, submit button or clear button; media-related UI objects such as pictures, slide shows, video or LBS maps; text-related UI objects such as buttons or paragraphs; and variables, including but not limited to time, date and audio mute control.

Events menu **301f** includes selections for defining child objects, mouse events, animations or timelines.

Pages menu **301g** includes selection for handling multi-page Applications, and may include selections to set a master page, delete, copy, add or go to Applications Pages.

Styles menu **301h** includes selections to handle styles, which are the underlying set of default appearance attributes or behaviors that define any object that is attached to a style. Styles are a convenient way for quickly creating complex objects, and for changing a whole collection of objects by just modifying their common style. Selections of Styles menu **301h** include, but not limited to, define, import, or modify a style, or apply a template. Help menu **301i** includes access a variety of help topics.

Tool bar **303** provides more direct access to some of the features of publisher interface **300** through a series of pull-down menus. Selections under tool bar **303** may include selections to:

control the look of publisher interface **300**, such as a Panel selection to control the for hiding or viewing various panels on publisher interface **300**;

control the layout being designed, such as an Insert Page selection to permit a user to insert and name pages;

control the functionality of publisher interface **300**, such as a Palettes selection to choose from a variety of specialized palettes, such as a View Palette for zooming and controlling the display of canvas **305**, a Command Palette of common commands, and Color and Shape Palettes;

place objects on canvas **305**, which may include selections such as: a Navigation selection to place navigational objects, such as widget or channel launch strips or selection lists), a Messages selection to place objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert, a Forms selection to place objects such as text fields or



US 9,471,287 B2

19

areas, check boxes, pull down menus, selection lists, and buttons, a Media selection to place pictures, slide shows, video or LBS maps, and a Shapes selection having a variety of drawing tools, a Text selection for placing text, a search response, or an RSS display, and Palettes.

In one embodiment, Tool bar **303** includes a series of pull-down menus that may include, but are not limited to, items from Menu bar **301** organized in the following pull-down menus: a Panel menu **303a**, an Insert Page menu **303b**, a Navigation menu **303c**, a Messages menu **303d**, a Forms menu **303e**, a Media menu **303f**, a Shapes menu **303g**, a Text menu **303h**, and a Palettes menu **301i**.

Panel menu **303a** permits a user of authoring platform **110** to change the appearance of interface **300** by, controlling which tools are on the interface or the size of canvas **305**. Insert Page menu **303b** permits a user of authoring platform **110** to open a new Application Page. Navigation menu **303c** displays a drop down menu of navigational-related objects such as a widget or channel launch strip or selection list. Messages menu **303d** displays a drop down menu of messaging-related objects such as multiple chat, video chat, phone or SMS lists or fields, and pop-up alerts. Forms menu **303e** displays a drop down menu of forms-related objects including, but not limited to, a text field, a text area, a check box, a drop down menu, a selection list, a submit button, and a clear button. Media menu **303f** displays a drop down menu of media-related objects including, but not limited to, a picture, slide show, video or LBS map. Shapes menu **303g** displays a drop down menu of draw tools, basic shapes, different types of lines and arrows and access to a shape library. Text menu **303h** displays a drop down menu of text-related objects, including but not limited to a text button, paragraph, search response, RSS display and variables such as time and date.

Palettes menu **301i** includes a selection of different palettes that can be moved about publisher interface **300**, where each palette has specialized commands for making adjustments or associations to objects easier. Palettes include, but are not limited to: a page view palette, to permit easy movement between Applications Pages; a view palette, to execute an Application or zoom or otherwise control the viewing of an Application; a commands palette having editing commands; a color palette for selection of object colors; and a shapes palette to facilitate drawing objects.

Layer inspector **307** permits a user of publisher interface **300** to navigate, select and manipulate UI objects on Applications Pages. Thus, for example, a Page/objects panel **307a** of layer inspector **307** has a listing that may be selected to choose an Applications Pages within an Application, and UI objects and styles within an Applications Page. An Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits selection of UI objects for operations to be performed on the objects.

Thus, for example, when objects from Menu bar **301** or Tool bar **303** are placed on canvas **305**, the name of the object appears in Page/objects panel **307a**. Page/objects panel **307a** includes a page display **307a1** and an objects display **307a2**. Page display **307a1** includes a pull down menu listing all Applications Pages of the Application, and objects display **307a2** includes a list of all objects in the Applications Page (that is, objects on canvas **305**).

In general, page/objects panel **307a** displays various associations with a UI object and permits various manipulations including, but not limited to, operations for parent and child objects that are assigned to a page, and operations for object styles, and permits navigating between page types

20

and object styles, such as switching between the master page and Application pages and deselecting object styles and alerts, opening an Edit Styles Dialog Box and deselecting any master, Application or alert page, or selecting an alert page and deselecting any Master Page or Application Page. A parent or child object can also be selected directly from the Canvas. In either case, the Resource Inspector can then be used for modifying any attribute of the selected object.

Examples of operations provided by page/objects panel **307a** on pages include, but are not limited to: importing from either a user's private page library or a public page library; deleting a page; inserting a new page, inheriting all the attributes of the Master Page, and placing the new page at any location in the Page List; editing the currently selected page, by working with an Edit Page Dialog Box. While editing all the functions of the Resource Inspector **309** are available, as described subsequently, but are not applied to the actual page until completing the editing process.

Examples of operations provided by of page/objects panel **307a** on objects, which may be user interface (UI) objects, include but are not limited to: changing the drawing order layer to: bring to the front, send to the back, bring to the front one layer, or send to the back one layer; hiding (and then reshowing) selected objects to show UI objects obstructed by other UI Objects, delete a selected UI Page Object, and editing the currently selected page, by working with a Edit Page Dialog Box.

Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits operations to be performed on objects, and is similar to panel **307a**. Examples of operations provided by object style panel **307b** include, but are not limited to: importing from either a user's private object library or a public object library; inserting a new object style, which can be inherited from a currently selected object, or from a previously defined style object; and editing a currently selected object style by working with an Edit Style Dialog Box.

Style attributes can be assigned many attributes, including the look, and behavior of any object that inherits these objects. In addition, List Layout Styles can be created or changed as required. A layout style can define a unbounded set of Complex List Layouts, including but not limited to: the number of lines per item in a list, the number of text and image elements and their location for each line for each item in the last, the color and font for each text element, and the vertical and horizontal offset for each image and text element.

Alerts Panel **307c** provides a way of providing alert pages, which can have many of the attributes of Application Pages, but they are only activated through an Event such as a user interaction, a network event, a timer event, or a system variable setting, and will be superimposed onto whatever is currently being displayed. Alert Pages all have transparent backgrounds, and they function as a template overlay, and can also have dynamic binding to real time content.

Resource inspector **309** is the primary panel for interactively working with UI objects that have been placed on the Canvas **305**. When a UI object is selected on Canvas **305**, a user of authoring platform **110** may associate properties of the selected object by entering or selecting from resource inspector **309**. In one embodiment, resource inspector **309** includes five tab selections: Setting Tab **309a**, Events Tab **309b**, Animation Tab **309c**, Color Tab **309d** which includes a color palette for selecting object colors, and Bindings Tab **309e**.

Settings Tab **309a** provides a dialog box for the basic configuration of the selected object including, but not lim-

## US 9,471,287 B2

21

ited to, name, size, location, navigation and visual settings. Depending upon the type of object, numerous other attributes could be settable. As an example, the Setting Tab for a Text Field may include dialog boxes to define the text field string, define the object style, set the font name, size and effects, set an object name, frame style, frame width, text attributes (text field, password field, numeric field, phone number, SMS number, URL request).

As an example of Setting Tab **309a**, FIG. 3B shows various selections including, but not limited to, setting **309a1** for the web page name, setting **309a2** for the page size, including selections for specific devices **130**, setting **309a3** indicating the width and height of the object, and setting **309a4** to select whether background audio is present and to select an audio file.

FIG. 3C illustrates an embodiment of the Events Tab **309b**, which includes all end user interactions and time based operations. The embodiment of Events Tab **309b** in FIG. 3C includes, for example and without limitation, an Events and Services **309b1**, Advanced Interactive Settings **309b2**, Mouse State **309b3**, Object Selected Audio Setting **309b4**, and Work with Child Objects and Mouse Overs button **309b5**.

Events and Services **309b1** lists events and services that may be applied to the selected objects. These include, but are not limited to, going to external web pages or other Applications pages, either as a new page or by launching a new window, executing an Application or JavaScript method, pausing or exiting, placing a phone call or SMS message, with or without single or multiple Player download, show launch strip, or go back to previous page. Examples of events and services include, but are not limited to those listed in Table II

TABLE II

Events and Services	
Goto External Web Page replacing Current Frame	ChoiceObject: Remove Icon from Launch Strip
Goto External Web Page Launched in a New Window	Goto a specific Internal Web Page with Alert. "Backend Synchronization"
Goto a specific Internal Web Page	Goto Widget Object
Goto the next Internal Web Page	Generate Alert. "With a Fire Event"
Goto External Web Page replacing the Top Frame	Send SMS Message from Linked Text Field
Execute JavaScript Method	Toggle Alert. "Display OnFocus, Hide OffFocus"
Pause/Resume Page Timeout	Execute an Application with Alert. "With a Fire Event"
Execute an Application	Goto Logical First Page
Goto a specific Internal Web Page with setting starting slide	Generate Alert with Backend Synchronization
Exit Application	Send SMS Message with Share (Player Download)
Exit Player	Place PhoneCall from linked Text Field with Share (Player Download)
Place PhoneCall from linked Text Field	Send IM Alert from linked Text Field or Text Area
Text Field/Area: Send String on FIRE	Set and Goto Starting Page
ChoiceObject: Add Icon to Launch Strip	Populate Image
Text Field/Area: Send String on FIRE or Numeric Keys	Preferred Launch Strip

Advanced Interactive Settings **309b2** include Scroll Activation Enabled, Timeline Entry Suppressed, Enable Server Listener, Submit Form, Toggle Children on FIRE, and Hide Non-related Children, Mouse State **309b3** selections are Selected or Fire. When Mouse State Selected is chosen,

22

Object Selected Audio Setting **309b4** of Inactive, Play Once, Loop, and other responses are presented. When Mouse State Fire is chosen, Object Selected Audio Setting **309b4** is replaced with FIRE Audi Setting, with appropriate choices presented.

When Work with Child Objects and Mouse Overs button **309b5** is selected, a Child Object Mode box pops up, allowing a user to create a child object with shortcut to Menu bar **301** actions that may be used define child objects.

FIG. 3D illustrates one embodiment of an Animation Tab **309c**, which includes all animations and timelines. The Color Tab includes all the possible color attributes, which may vary significantly by object type.

Animation Tab **309c** includes settings involved in animation and timelines that may be associated with objects. One embodiment of Animation Tab **309c** is shown, without limitation, in FIG. 3D, and is described, in Rempell ("Rempell").

A Color Tab **309d** includes a color palette for selecting object colors.

Bindings Tab **309e** is where web component operations are defined and dynamic binding settings are assigned. Thus, for example, a UI object is selected from canvas **305**, and a web component may be selected and configured from the bindings tab. When the user's work is saved, binding information is associated with the UI object that will appear on screen **137**.

FIG. 3E illustrates one embodiment of Bindings Tab and includes, without limitation, the following portions: Web Component and Web Services Operations **309e1**, Attributes Exposed list **309e2**, panel **309e3** which includes dynamic binding of server-side data base values to attributes for the selected object, Default Attribute Value **309e4**, Database Name **309e5**, Table Name **309e6**, Field Name **309e7**, Channel Name **309e8**, Channel Feed **309e9**, Operation **309e10**, Select Link **309e11**, and Link Set checkbox **309e12**.

Web Component and Web Services Operations **309e1** includes web components that may be added, edited or removed from a selected object. Since multiple web components can be added to the same object, any combination of mash-ups of 3rd party web services is possible. When the "Add" button of Web Component and Web Services Operations **309e1** is selected, a pop-up menu **319**, as shown in FIG. 3F, appears on publisher interface **300**. Pop-up menu **319** includes, but is not limited to, the options of: Select a Web Component **319a**; Select Results Page **319b**; Activation Options **319c**; Generate UI Objects **319d**; and Share Web Component **319e**.

The Select a Web Component **319a** portion presents a list of web components. As discussed herein, the web components are registered and are obtained from web component registry **220**.

Select Results Page **319b** is used to have the input and output on different pages—that is, when the Results page is different from Input page. The default selected results page is either the current page, or, if there are both inputs and outputs, it will be set provisionally to the next page in the current page order, if one exists.

Activation Options **319c** include, if there are no Input UI Objects, a choice to either "Preload" the web component, similar to how dynamic binding, or have the web component executed when the "Results" page is viewed by the consumer.

Generate UI Objects **319d**, if selected, will automatically generate the UI objects. If not selected, then the author will bind the Web Component Inputs and Results to previously created UI Objects.

US 9,471,287 B2

23

Share Web Component **319e** is available and will become selected under the following conditions: 1) Web Component is Selected which already has been used by the current Application; or 2) the current Input page is also a "Result" page for that Web component. This permits the user of device **130**, after viewing the results, to extend the Web Component allowing the user to make additional queries against the same Web Component. Examples of this include, but are not limited to, interactive panning and zooming for a Mapping Application, or additional and or refined searches for a Search Application.

Dynamic Binding permits the binding of real time data, that could either reside in a 3rd party server-side data base, or in the database maintained by Feed Collector **1010** for aggregating live RSS feeds, as described subsequently with reference to FIG. **10**.

Referring again to FIG. **3E**, Attributes Exposed list **309e2** are the attributes available for the selected object that can be defined in real time through dynamic binding.

Panel **309e3** exposes all the fields and tables associated with registered server-side data bases. In one embodiment, the user would select an attribute from the "Attributes Exposed List" and then select a data base, table and field to define the real time binding process. The final step is to define the record. If the Feed Collector data base is selected, for example, then the RSS "Channel Name" and the "Channel Feed" drop down menus will be available for symbolically selected the record. For other data bases the RSS "Channel Name" and the "Channel Feed" drop down menus are replaced by a "Record ID" text field.

Default Attribute Value **309e4** indicates the currently defined value for the selected attribute. It will be overridden in real time based on the dynamic linkage setting.

Database Name **309e5** indicates which server side data base is currently selected.

Table Name **309e6** indicates which table of the server side data base is currently selected.

Field Name **309e7**, indicates which field form the selected table of the server side data base is currently selected.

Channel Name **309e8** indicates a list of all the RSS feeds currently supported by the Feed Collector. This may be replaced by "Record ID" if a data base other than the Feed Collector **1010** is selected.

Channel Feed **309e9** indicates the particular RSS feed for the selected RSS Channel. Feed Collector **1010** may maintain multiple feeds for each RSS channel.

Operation **309e10**, as a default operation, replaces the default attribute value with the real time value. In other embodiments this operation could be append, add, subtract, multiply or divide.

Select Link **309e11** a button that, when pressed, creates the dynamic binding. Touching the "Select Link" will cause the current data base selections to begin the blink is some manner, and the "Select Link" will change to "Create Link". The user could still change the data base and attribute choices. Touching the "Create Link" will set the "Link Set" checkbox and the "Create Link" will be replaced by "Delete Link" if the user wishes to subsequently remove the link. When the application is saved, the current active links are used to create the SPDL.

Link Set checkbox **309e12** indicates that a link is currently active.

An example of the design of a display is shown in FIGS. **4A** and **4B** according to the system **100**, where FIG. **4A** shows publisher interface **300** having a layout **410** on canvas **305**, and FIG. **4B** shows a device **130** having the resulting layout **420** on screen **137**. Thus, for example, authoring platform

24

**110** is used to design layout **410**. Authoring platform **110** then generates an Application and a Player specific to device **130** of FIG. **4B**. The Application and Player are thus used by device **130** to produce layout **420** on screen **137**.

As illustrated in FIG. **4A**, a user has placed the following on canvas **305** to generate layout **410**: text and background designs **411**, a first text input box **413**, a second text input box **415**, and a button **417**. As an example which is not meant to limit the scope of the present invention, layout **410** is screen prompts a user to enter a user name in box **413** and a password in box **415**, and enter the information by clicking on button **417**.

In one embodiment, all UI objects are initially rendered as Java objects on canvas **305**. When the Application is saved, the UI objects are transformed into the PDL, as described subsequently.

Thus, for example, layout **410** may be produced by the user of authoring platform **110** selecting and placing a first Text Field as box **413** then using the Resource Inspector **309** portion of interface **300** to define its attributes.

Device User Experience

Systems **100** and **200** provide the ability for a very large number of different types of user experiences. Some of these are a direct result of the ability of authoring platform **110** to bind UI objects to components of web services. The following description is illustrative of some of the many types of experiences of using a device **130** as part of system **100** or **200**.

Device **130** may have a one or more of a very powerful and broad set of extensible navigation objects, as well as object- and pointer-navigation options to make it easy to provide a small mobile device screen **137** with content and to navigate easily among page views, between Applications, or within objects in a single page view of an Application.

Navigation objects include various types of launch strips, various intelligent and user-friendly text fields and scrolling text boxes, powerful graphical complex lists, as well as Desktop-level business forms. In fact, every type of object can be used for navigation by assigning a navigation event to it. The authoring tool offers a list of navigation object templates, which then can be modified in numerous ways. Launch Strips and Graphical List Templates Launch Strips Launch strips may be designed by the user of authoring platform **110** with almost no restrictions. They can be stationary or appear on command from any edge of the device, their size, style, audio feedback, and animations can be freely defined to create highly compelling experiences.

FIG. **5** shows a display **500** of launch strips which may be on displayed canvas **305** or on screen **137** of device **130** having the proper Player and Application. Display **501** includes a portal-type Launch Strip **501** and a channel-type Launch Strip **502**, either one of which may be included for navigating the Application.

Launch Strip **501** includes UI objects **501a**, **501b**, **501c**, **501d**, and **501e** that that becomes visible from the left edge of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector **309** with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. When the Applications Page, having been saved by authoring platform **110** and transferred to display **130**, is executed on device **130**, a user of the device may easily navigate the Application.

Launch Strip **502** includes UI objects **502b**, **502c**, **502d**, and **503e** that that becomes visible from the bottom of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector

## US 9,471,287 B2

25

309 with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. Launch Strip 502 also includes UI objects 502a and 503g, which include the graphic of arrows, and which provide access to additional navigation objects (not shown) when selected by a user of device 130. Launch strip 502 may also include sound effects for each channel when being selected, as well as popup bubble help.

Additional navigational features are illustrated in FIG. 6A as a display of a Channel Selection List 601a, in FIG. 6B as a display of a Widget Selection List 601b, and in FIG. 6C as a display of a Phone List 601c. Lists 601a, 601b, and 601c may be displayed on canvas 305 or on screen 137 of device 130 having the proper Player and Application. As illustrated, graphical lists 601a, 601b, and 601c may contain items with many possible text and image elements. Each element can be defined at authoring time and/or populated dynamically through one or more Web Service 250 or API. Assignable Navigation Events. All objects, and/or all elements within an object, can be assigned navigation events that can be extended to registered web services or APIs. For example, a Rolodex-type of navigation event can dynamically set the starting slide of the targeted page view (or the starting view of a targeted Application).

In the embodiment of FIGS. 6A, 6B, and 6C, each list 601a, 601b, and 601c has several individual entries that are each linked to specific actions. Thus Channel Selection List 601a shows three objects, each dynamically linked to a web service (ESPN, SF 49ers, and Netflix) each providing a link to purchase or obtain items from the Internet. Widget Selection List 601b includes several objects presenting different widgets for selecting. Phone List 601c includes a list phone number objects of names that, when selected by a user of device 130 cause the number to be dialed. Entries in Phone List 601c may be generated automatically from either the user's contact list that is resident on the device, or through a dynamic link to any of user's chosen server-side facilities such as Microsoft Outlook, Google Mail, etc. In one embodiment, Phone List 601c may be generated automatically using a web component assigned to the Application, which would automatically perform those functions.

In another embodiment, authoring platform 110 allows a navigation selection of objects with a Joy Stick and/or Cursor Keys in all 4 directions. When within a complex object the navigation system automatically adopts to the navigation needs for that object. For coordinate sensitive objects such as geographical information services (GIS) and location-based services (LBS) or virtual tours a soft cursor appears. For Lists, scrolling text areas and chats, Launch strips, and slide shows the navigation process permits intuitive selection of elements within the object. Scroll bars and elevators are optionally available for feedback. If the device has a pointing mechanism then scroll bars are active and simulate the desktop experience.

Personalization and Temporal Adoption

System 100 and 200 permit for the personalization of device 130 by a variety of means. Specifically, what is displayed on screen 137 may depend on either adoption or customization. Adoption refers to the selection of choices, navigation options, etc. are based on user usage patterns. Temporal adoption permits the skins, choices, layouts, content, widgets, etc. to be further influenced by location (for example home, work or traveling) and time of day (including season and day of week). Customization refers to user selectable skins, choices, layouts, dynamic content, widgets, etc. that are available either through a customization on the

26

phone or one that is on the desktop but dynamically linked to the user's other internet connected devices.

To support many personalization functions there must be a convenient method for maintaining, both within a user's session, and between sessions, memory about various user choices and events. Both utilizing a persistent storage mechanism on the device, or a database for user profiles on a server, may be employed.

FIG. 7 shows a display 700 of a mash-up which may be on displayed canvas 305 or on screen 137 of device 130 having the proper Player and Application. Display 700 includes several object 701 that have been dynamically bound, including an indication of time 701a, an indication of unread text messages 701b, an RSS news feed 701c (including 2 "ESPN Top Stories" 701c1 and 701c2), components 701d from two Web Services—a weather report ("The Weather Channel"), and a traffic report 701e ("TRAFFIC.COM").

In assembling the information of display 700, device 130 is aware of the time and location of the device—in this example the display is for a workday when a user wakes. Device 130 has been customized so that on a work day morning the user wishes to receive the displayed information. Thus in the morning, any messages received overnight would be flagged, the user's favorite RSS sports feeds would be visible, today's weather forecast would be available, and the current traffic conditions between the user's home and office would be graphically depicted. User personalization settings may be maintained as persistent storage on device 130 when appropriate, or in a user profile which is maintained and updated in real-time in a server-side data base. Push Capable Systems

In another embodiment system 100 or 200 is a push-capable system. As an example, of such systems, short codes may be applied to cereal boxes and beverage containers, and SMS text fields can be applied to promotional websites. In either case, a user of device 130 can text the short code or text field to an SMS server, which then serves the appropriate Application link back to device 130.

FIG. 8 is a schematic of an embodiment of a push enabled system 800. System 800 is generally similar to system 100 or 200. Device 130 is shown as part of a schematic of a push capable system 800 in FIG. 8. System 800 includes a website system 801 hosting a website 801, a server 803 and a content server 805. System 801 is connected to servers 803 and/or 805 through the Internet. Server 803 is generally similar to server 120, servers 805 is generally similar to server 140.

In one embodiment, a user sets up a weekly SMS update from website system 801. System 801 provides user information to server 803, which is an SMS server, when an update is ready for delivery. Server 803 provides device 130 with an SMS indication that the subscribed information is available and queries the user to see if they wish to receive the update. Website 801 also provides content server 805 with the content of the update. When a user of device 130 responds to the SMS query, the response is provided to content server 805, which provides device 130 with updates including the subscribed content.

In an alternative embodiment of system 800, server 803 broadcasts alerts to one or more devices 130, such as a logical group of devices. The user is notified in real-time of the pending alert, and can view and interact with the message without interrupting the current Application.

FIG. 9 is a schematic of an alternative embodiment of a push enabled system 900. System 900 is generally similar to system 100, 200, or 800. In system 900 a user requests information using an SMS code, which is delivered to device

US 9,471,287 B2

27

130. System 900 includes a promotional code 901, a third-party server 903, and content server 805. Server 803 is connected to servers 803 and/or 805 through the Internet, and is generally similar to server 120.

A promotional code 901 is provided to a user of device 130, for example and without limitation, on print media, such as on a cereal box. The use of device 130 sends the code server 903. Server 903 then notifies server 805 to provide certain information to device 130. Server 805 then provides device 130 with the requested information.

Device Routines

Device routines 114 may include, but are not limited to: an authoring tool SDK for custom code development including full set of Java APIs to make it easy to add extensions and functionality to mobile Applications and tie Applications to back-end databases through the content server 140; an expanding set of web services 250 available through the authoring tool SDK; a web services interface to SOAP/XML enabled web services; and an RSS/Atom and RDF feed collector 1010 and content gateway 1130.

Authoring Tool SDK for Custom Code Development Including Full Set of Java APIs

In one embodiment, authoring platform 110 SDK is compatible for working with various integrated development environments (IDE) and popular plug ins such as J2ME Polish. In one embodiment the SDK would be another plug in to these IDEs. A large and powerful set of APIs and interfaces are thus available through the SDK to permit the seamless extension of any Application to back end business logic, web services, etc. These interfaces and APIs may also support listeners and player-side object operations.

There is a large set of listeners that expose both player-side events and dynamically linked server side data base events. Some examples of player side events are: player-side time based event, a page entry event, player-side user interactions and player-side object status. Examples of server-side data base events are when a particular set of linked data base field values change, or some filed value exceeds a certain limit, etc.

A superset of all authoring tool functionality is available through APIs for layer-side object operations. These include, but are not limited to: page view level APIs for inserting, replacing, and or modifying any page object; Object Level APIs for modifying any attribute of existing objects, adding definitions to attributes, and adding, hiding or replacing any object.

Authoring Tool SDK Available Web Services

The APIs permit, without limit, respond, with or without relying on back-end business logic, that is, logic that what an enterprise has developed for their business, to any player-side event or server-side dynamically linked data-base, incorporating any open 3rd party web service(s) into the response.

RSS/Atom and RDF Feed Conversion Web Service

FIG. 10 is a schematic of one embodiment a system 1000 having a feed collector 1010. System 1000 is generally similar to system 100, 200, 800, or 900. Feed collector 1010 is a server side component of system 100 that collects RSS, ATOM and RDF format feeds from various sources 1001 and aggregates them into a database 1022 for use by the Applications built using authoring platform 110.

Feed collector 1010 is a standard XML DOM data extraction process, and includes Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, and Custom Populator Rule 1016, DOM XML Parsers 1011, 1015, and 1017, Feed Processed Data Writer 1018, Custom Rule

28

Based Field Extraction 1019, Rule-based Field Extraction 1020, Channel Data Controller 1021, and Database 1022.

The feed collector is primarily driven by two sets of parameters: one is the database schema (written as SQL DDL) which defines the tables in the database, as well as parameters for each of the feeds to be examined. The other is the feed collection rules, written in XML, which can be used to customize the information that is extracted from the feeds. Each of the feeds is collected at intervals specified by the feed parameter set in the SQL DDL.

Feed collector 1010 accepts information from ATOM, RDF or RSS feed sources 1001. Using a rules-based populator, any of these feeds can be logically parsed, with any type of data extraction methodology, either by using supplied rules, or by the author defining their own custom extraction rule. The rules are used by the parser to parse from the feed sources, and the custom rule base field extraction replaces the default rules and assembles the parsed information into the database.

In particular, Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, Custom Populator Rule 1016, and DOM XML Parsers 1011, 1015, and 1017 are parse information from the feeds 1001, and Feed Processed Data Writer 1018, Custom Rule Based Field Extraction 1019, Rule-based Field Extraction 1020, and Channel Data Controller 1021, supply the content of the feeds in Database 1022, which is accessible through content server 140.

FIG. 11 is a schematic of an embodiment of a system 1100 having a Mobile Content Gateway 1130. System 1100 is generally similar to system 100, 200, 800, 900, or 1000. System 1100 includes an SDK 1131, feed collector 1010, database listener 1133, transaction server 1134, custom code 1135 generated from the SDK, Java APIs, Web Services 1137, and PDL snippets compacted objects 1139. System 1100 accepts input from Back End Java Code Developer 1120 and SOAP XML from Web Services 1110, and provides dynamic content to server 140 and Players to devices 130.

In one embodiment authoring platform 110 produces a Server-side PDL (SPDL) at authoring time. The SPDL resides in server 120 and provides a logical link between the Application's UI attributes and dynamic content in database 1022. When a user of device 130 requests dynamic information, server 120 uses the SPDL to determine the link required to access the requested content.

In another embodiment Web Services 1137 interface directly with 3rd party Web Services 1110, using SOAP, REST, JAVA, JavaScript, or any other interface for dynamically updating the attributes of the Application's UI objects. XSP Web Pages as a Web Service

In one embodiment, a PDL for a page is embedded within an HTML shell, forming one XSP page. The process of forming XSP includes compressing the description of the page and then embedding the page within an HTML shell.

In another embodiment, a PDL, which contains many individual page definitions, is split into separate library objects on the server, so that each page can be presented as a PDL as part of a Web Service.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java), and The code and data have been reduced by 4 to 10 times.

Compression has two distinct phases. The first takes advantage of how the primitive representations had been assembled, while the second utilizes standard LZ encoding.

US 9,471,287 B2

29

The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

One embodiment for compacting data that may be used is described in Rempell. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

In Rempell, a process for compacting a "database" (that is, generating a compact PDL) is described, wherein data objects, including but not limited to, multi media objects such as colors, fonts, images, sound clips, URLs, threads, and video, including multi level animation, transformation, and time line are compacted. As an extension to Rempell in all cases these objects are reduced and transformed to Boolean, integer and string arrays.

The compression technique involves storing data in the smallest arrays necessary to compactly store web page information. The technique also includes an advanced form of delta compression that reduces integers so that they can be stored in a single byte, a as high water marks.

Thus, for example, the high water mark for different types of data comprising specific web site settings are stored in a header record as Boolean and integer variables and URL and color objects. Data that defines web page, paragraph, text button, and image style and text button, image and paragraph high watermark settings can be stored in one-dimensional arrays as Boolean, integer and string variables and URL, font, image or thread objects at. The URL, color, font, image and thread objects can also be created as required.

Data that defines text button, image, paragraph, or other parent objects and paragraph line high watermark settings can be stored in two-dimensional arrays (by web page and by object number) as Boolean, integer, string, floating point variables and URLs. Again, the URL, color, font, image, audio clip, video clip, text area and thread objects can also be created as required. Data that defines a paragraph line and paragraph line segment high watermarks can be stored in three-dimensional arrays (by web page, by paragraph number, and by line number) as Boolean, integer or string variables. Again, the URL, color or font objects can be created as required. Data that defines a paragraph line segment can be stored into four-dimensional arrays (by web page, by paragraph number, by line number and by line number segment) as Boolean, integer or string variables or URL, color and font objects.

As a data field is added, changed or deleted, a determination is made at on whether a value for a given high watermark needs to be changed. If so, it is updated. As a specific method in the build engine is called, a determination is made on whether a feature flag needs to be set. For example, if a particular JAVA method is called, which requires an instance of a certain JAVA Class to be executed by the run time engine, then that JAVA Class is flagged, as well as any supporting methods, variables and/or object definitions.

In one implementation, the header record, the style record, the web page record, and the object records, are carefully defined in a specific order, written in that order, and explicitly cast by object type when read by the run time engine. Exception handling can be implemented to recover from any errors. This helps assure that data integrity is maintained throughout the build and run time processes.

Also described in Rempell is the "run generation process." This is equivalent generating a Player in the present application. This process starts when the build process detects that the user is finished defining the web site (user has saved the web site and invokes the run generation

30

process), and concludes with the actual uploading of all the necessary web site run time files to the user's server.

In one embodiment, the PDL includes a first record, a "Header" record, which contains can include the following information:

- 1: A file format version number, used for upgrading database in future releases.
- 2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user.
- 3: Whether the Application is a web site.
- 4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.
- 5: Web page and styles high watermarks.
- 6: The Websitename.

As new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the PDL, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

The settings for all paragraph, text button and image styles are then written as a style record based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist.

The body of the PDL is then written. All Boolean values are written inside a four-dimensional loop. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects ((i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of

US 9,471,287 B2

31

line segments per paragraph line and contains the values used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment).

All integer values are written inside a four-dimensional loop. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

Multimedia objects are written inside a two-dimensional loop. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a web browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other objects on a web page.

String records are written inside a four-dimensional loop. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The third loop contains values for paragraph line related data, and the innermost loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized.

Single and double floating-point, and long integer records are written inside a two-dimensional loop. The outer loop may be empty. The inner loop contains mathematical values required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

In one embodiment, a versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation.

All external image, video and audio files are resolved. The external references can be copied to designated directories, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries are either installed on the local system or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code. Finally, the run time engine for the web site is created. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file.

Next, an HTML Shell File (HSF) is constructed. The first step of this process is to determine whether the dynamic web page and object resizing is desired by testing the Application setting. If the Application was a web page, and thus requir-

32

ing dynamic web page and object resizing, virtual screen resolution settings are placed in an appropriate HTML compliant string. If the Application is a banner or other customized Application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values.

An analysis is made for the background definition for the first internal web page. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the web browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a web browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the web browser. More specifically, if the Application required dynamic web page and object resizing then JavaScript and HTML compliant strings are generated so that, when interpreted by the web browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated in order to enable JavaScript to SRS applet communication. In one implementation, the code is generated by performing the following functions:

- 1: Determine the current web browser type.
- 2: Load the SRS from either a JAR or CAB File, based on web browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.
- 5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated that perform the following steps at the time the HSF is initialized by the web browser:

- 1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.
- 2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the PDL.
- 3: Generate an HTML complaint string, dependent upon the type of web browser, which causes the current web browser to load either the JAR or the CAB File(s).
- 4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the web browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

The writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "Website-name".html file is created.

In one embodiment, the processes for creating the CAB and JAR Files is as follows. The image objects, if any, which were defined on the first internal web page are analyzed. If they are set to draw immediately upon the loading of the first

US 9,471,287 B2

33

web page, then they are flagged for compression and inclusion in the CAB and JAR Files. The feature flags are analyzed to determine which JAVA classes have been compiled. These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "Website-name".class, customized run time engine file, and the "Website-name".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "Website-name".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "Website-name".bat and "Website-namelib".bat files are written. The "Website-name".bat and "Website-namelib".bat files are then executed under DOS, creating compressed "Website-name".cab and "Website-namelib".cab files and compressed "Website-name".jar and "Website-namelib".jar files. The HTML Shell File and the JAR and CAB files are then, either as an automatic process, or manually, uploaded to the user's web site. This completes the production of an XSP page that may be accessed through a web browser.

#### Displaying Content on a Device Decompression Management

Authoring platform 110 uses compaction to transform the code and data in an intelligent way while preserving all of the original classes, methods and attributes. This requires both an intelligent server engine and client (handset) Player, both of which fully understand what the data means and how it will be used.

The compaction technology described above includes transformation algorithms that deconstruct the logic and data into their most primitive representations, and then reassembles them in a way that can be optimally digested by further compression processing. This reassembled set of primitive representations defines the PDL of authoring platform 110.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java). The data is then compressed by first taking advantage of how the primitive representations had been assembled, and then by utilizing standard LZ encoding. The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

The Player, when preparing a page view for execution, decompresses and then regenerate the original objects, but this time in compliance with the programming APIs of device 130. Specifically, device 130 operates on compacted image pages, one at a time. The cache manager retrieves, decompresses, and reassembles the compacted page images into device objects, which are then interpreted by device 130 for display on screen 137.

#### Response Director

In one embodiment, system 100 includes a Response Director, which determines a user's handset, fetches the correct Application from different databases, and delivers a respective highly compressed Application in a PDL format over the air (OTA).

34

In one embodiment, the Response Director operates on a network connected computer to provide the correct Player to a given device based on the information the device sent to it. As an example, this may occur when a device user enters their phone number into some call-to-action web page. The response director is called and sends an SMS message to the device, which responds, beginning the recognition process.

FIG. 12 illustrates one embodiment of a system 1200 that includes a response director 210, a user agent database 1201, an IP address database 1203, and a file database 1205. System 1200 is generally similar to system 100, 200, 800, 900, 1000, or 1100.

Databases 1201, 1203, and 1205 may reside on server 120, 210, or any computer system in communication with response director 210. System 1200, any mobile device can be serviced, and the most appropriate Application for the device will be delivered to the device, based on the characteristics of the device.

User agent database 1201 includes user agent information regarding individual devices 130 that are used to identify the operating system on the device. IP address database 1203 identifies the carrier/operator of each device 130. File database 1205 includes data files that may operate on each device 130.

The following is an illustrative example of the operation of response director 210. First, a device 1300 generates an SMS message, which automatically sends an http:// stream that includes handset information and its phone number to response director 210. Response director 210 then looks at a field in the http header (which includes the user agent and IP address) that identifies the web browser (i.e., the "User Agent"). The User Agent prompts a database lookup in user agent database 1201 which returns data including, but not limited to, make, model, attributes, MIDP 1.0 MIDP 2.0, WAP and distinguishes the same models from different countries. A lookup of the IP address in IP address 1203 identifies the carrier/operator.

File database 1205 contains data types, which may include as jad1, jad2, html, wml/wap2, or other data types, appropriate for each device 130. A list of available Applications are returned to a decision tree, which then returns, to device 130, the Application that is appropriate for the respective device. For each file type, there is an attributes list (e.g., streaming video, embedded video, streaming audio, etc.) to provide enough information to determine what to send to the handset.

Response director 210 generates or updates an html or jad file populating this text file with the necessary device and network dependent parameters, including the Application dependent parameters, and then generate, for example, a CAB or JAD file which contains the necessary Player for that device. For example, the jad file could contain the operator or device type or extended device-specific functions that the player would then become aware of.

If there is an Application that has a data type that device 130 cannot support, for example, video, response director 210 sends an alternative Application to the handset, for example one that has a slide show instead. If the device cannot support a slide show, an Application might have text and images and display a message that indicates it does not support video.

Another powerful feature of response director 210 is its exposed API from the decision tree that permits the overriding of the default output of the decision tree by solution providers. These solution providers are often licensees who want to further refine the fulfillment of Applications and Players to specific devices beyond what the default algo-



US 9,471,287 B2

35

rithms provide. Solution providers may be given a choice of Applications and then can decide to use the defaults or force other Applications.

Authoring platform 110 automatically scales Applications at publishing time to various form factors to reduce the amount of fragmentation among devices, and the Response Director serves the appropriately scaled version to the device. For example, a QVGA Application will automatically scale to the QCIF form factor. This is important because one of the most visible forms of fragmentation resides in the various form factors of wireless, and particularly mobile, devices, which range from 128×128, 176×208, 240×260, 220×220, and many other customized sizes in between.

FIG. 13 is a schematic of an embodiment of a system 1300. System 1300 is generally similar to system 1200. System 1300 is an overview of the entire Player fulfillment process, starting with the generation of players during the player build process.

System 1300 includes response director 210, a device characteristics operator and local database 1301, a player profile database 1303 and a player build process 1305, which may be authoring platform 110.

As an example of system 1300, when response director 210 receives an SMS message from device 130, the response director identifies the device characteristics operator and locale from database 1301 and a Player URL from database 1303 and provides the appropriate Player to the device.

In another embodiment, Player P extend the power of response director 210 by adapting the Application to the resources and limitations of any particular device. Some of these areas of adaptation include the speed of the device's microprocessor, the presence of device resources such as cameras and touch screens. Another area of adaptation is directed to heap, record store and file system memory constraints. In one embodiment, the Player will automatically throttle down an animation to the frame rate that the device can handle so that the best possible user experience is preserved. Other extensions include device specific facilities such as location awareness, advanced touch screen interactions, push extensions, access to advanced phone facilities, and many others

#### Memory Management

In one embodiment, Player P includes a logical page virtual memory manager. This architecture requires no supporting hardware and works efficiently with constrained devices. All page view images, which could span multiple Applications, are placed in a table as highly compacted and compressed code. A typical page view will range from 500 bytes up to about 1,500 bytes. (See, for example, the Rempell patent) When rolled into the heap and instantiated this code increases to the more typical 50,000 up to 250,000 bytes. Additional alert pages may also be rolled into the heap and superimposed on the current page view. Any changes to any page currently downloaded are placed in a highly compact change vector for each page, and rolled out when the page is discarded. Note that whenever an Application is visited that had previously been placed in virtual memory the Server is interrogated to see if a more current version is available, and, if so, downloads it. This means that Application logic can be changed in real-time and the results immediately available to mobile devices.

To operate efficiently with the bandwidth constraints of mobile devices, authoring platform 110 may also utilize anticipatory streaming and multi-level caching. Anticipatory streaming includes multiple asynchronous threads and IO request queues. In this process, the current Application is

36

scanned to determine if there is content that is likely to be required in as-yet untouched page views. Anticipatory streaming also looks for mapping Applications, where the user may zoom or pan next so that map content is retrieved prior to the user requesting it. For mapping applications, anticipatory streaming downloads a map whose size is greater than the map portal size on the device and centered within the portal. Any pan operation will anticipatory stream a section of the map to extend the view in the direction of the pan while, as a lower priority, bring down the next and prior zoom levels for this new geography. Zooming will always anticipatory stream the next zoom level up and down.

Multi-level caching determines the handset's heap through an API, and also looks at the record store to see how much memory is resident. This content is placed in record store and/or the file system, and may, if there is available heap, also place the content there as well. Multi-level caching permits the management of memory such that mobile systems best use limited memory resources. Multi-level caching is a memory management system with results similar to embedding, without the overhead of instantiating the content. In other words, with multi-level caching, handset users get an "embedded" performance without the embedded download. Note that when content is flagged as cacheable and is placed in persistent storage, a digital rights management (DRM) solution will be used.

One embodiment of each of the methods described herein is in the form of a computer program that executes on a processing system. Thus, as will be appreciated by those skilled in the art, embodiments of the present invention may be embodied as a method, an apparatus such as a special purpose apparatus, an apparatus such as a data processing system, or a carrier medium, e.g., a computer program product. The carrier medium carries one or more computer readable code segments for controlling a processing system to implement a method. Accordingly, aspects of the present invention may take the form of a method, an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. Furthermore, the present invention may take the form of carrier medium (e.g., a computer program product on a computer-readable storage medium) carrying computer-readable program code segments embodied in the medium. Any suitable computer readable medium may be used including a magnetic storage device such as a diskette or a hard disk, or an optical storage device such as a CD-ROM.

It will be understood that the steps of methods discussed are performed in one embodiment by an appropriate processor (or processors) of a processing (i.e., computer) system executing instructions (code segments) stored in storage. It will also be understood that the invention is not limited to any particular implementation or programming technique and that the invention may be implemented using any appropriate techniques for implementing the functionality described herein. The invention is not limited to any particular programming language or operating system. It should thus be appreciated that although the coding for programming devices has not been discussed in detail, the invention is not limited to a specific coding method. Furthermore, the invention is not limited to any one type of network architecture and method of encapsulation, and thus may be utilized in conjunction with one or a combination of other network architectures/protocols.

Reference throughout this specification to "one embodiment," "an embodiment," or "certain embodiments" means that a particular feature, structure or characteristic described

US 9,471,287 B2

37

in connection with the embodiment is included in at least one embodiment of the present invention. Thus, appearances of the phrases “in one embodiment,” “in an embodiment,” or “in certain embodiments” in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures or characteristics may be combined in any suitable manner, as would be apparent to one of ordinary skill in the art from this disclosure, in one or more embodiments.

Throughout this specification, the term “comprising” shall be synonymous with “including,” “containing,” or “characterized by,” is inclusive or open-ended and does not exclude additional, unrecited elements or method steps. “Comprising” is a term of art which means that the named elements are essential, but other elements may be added and still form a construct within the scope of the statement. “Comprising” leaves open for the inclusion of unspecified ingredients even in major amounts.

Similarly, it should be appreciated that in the above description of exemplary embodiments, various features of the invention are sometimes grouped together in a single embodiment, figure, or description thereof for the purpose of streamlining the disclosure and aiding in the understanding of one or more of the various inventive aspects. This method of disclosure, however, is not to be interpreted as reflecting an intention that the claimed invention requires more features than are expressly recited in each claim. Rather, as the following claims reflect, inventive aspects lie in less than all features of a single foregoing disclosed embodiment, and the invention may include any of the different combinations embodied herein. Thus, the following claims are hereby expressly incorporated into this Mode(s) for Carrying Out the Invention, with each claim standing on its own as a separate embodiment of this invention.

Thus, while there has been described what is believed to be the preferred embodiments of the invention, those skilled in the art will recognize that other and further modifications may be made thereto without departing from the spirit of the invention, and it is intended to claim all such changes and modifications as fall within the scope of the invention. For example, any formulas given above are merely representative of procedures that may be used. Functionality may be added or deleted from the block diagrams and operations may be interchanged among functional blocks. Steps may be added or deleted to methods described within the scope of the present invention.

We claim:

1. A system for generating code to provide content on a display of a device, said system comprising:  
computer memory storing a registry of:

a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, where each symbolic name has an associated data format class type corresponding to a subclass of User Interface (UI) objects that support the data format type of the symbolic name, and has a preferred UI object, and

b) an address of the web service;  
an authoring tool configured to:

define a (UI) object for presentation on the display, where said defined UI object corresponds to a web component included in said registry selected from a group consisting of an input of the web service

38

and an output of the web service, where each defined UI object is either: 1) selected by a user of the authoring tool; or 2) automatically selected by the system as the preferred UI object corresponding to the symbolic name of the web component selected by the user of the authoring tool,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object, where the selected symbolic name is only available to UI objects that support the defined data format associated with that symbolic name, and

produce an Application including the selected symbolic name of the defined UI object, where said Application is a device-independent code; and

a Player, where said Player is a device-dependent code, wherein, when the Application and Player are provided to the device and executed on the device, and when the user of the device provides one or more input values associated with an input symbolic name to an input of the defined UI object,

1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,

2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,

3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for the display of the device to present an output value in the defined UI object.

2. The system of claim 1, where said registry includes definitions of input and output related to said web service.

3. The system of claim 1, where said web component is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

4. The system of claim 1, where said UI object is an input field for a chat.

5. The system of claim 1, where said UI object is an input field for a web service.

6. The system of claim 1, where said UI object is an input field usable to obtain said web component, where said input field includes a text field, a scrolling text box, a check box, a drop down-menu, a list menu, or a submit button.

7. The system of claim 1, where said web component is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

8. The system of claim 1, where said authoring tool is further configured to:

define a phone field or list; and  
generate code that, when executed on the device, allows a user to supply a phone number to said phone field or list.

9. The system of claim 1, where said authoring tool is further configured to:

define a SMS field or list; and  
generate code that, when executed on the device, allows a user to supply an SMS address to said SMS field or list.

US 9,471,287 B2

39

10. The system of claim 1,  
 where said code includes three or more codes, where one  
 of said three or more codes is device specific, and  
 where two of said three or more codes is device  
 independent. 5

11. The system of claim 1, where said code is provided  
 over said network.

12. The system of claim 1, wherein said defined UI object  
 corresponds to a widget.

13. The system of claim 1, where said Player is activated 10  
 and runs in a web browser.

14. The system of claim 1, where said Player is a native  
 program.

15. A method of displaying content on a display of a  
 device having a Player, where said Player is a device- 15  
 dependent code, said method comprising:  
 defining a user interface (UI) object for presentation on  
 the display, where said UI object corresponds to a web  
 component included in a registry of one or more web  
 components selected from a group consisting of an 20  
 input of a web service and an output of the web service,  
 where each web component includes a plurality of  
 symbolic names of inputs and outputs associated with  
 each web service, and where the registry includes: a)  
 symbolic names required for evoking one or more web 25  
 components each related to a set of inputs and outputs  
 of the web service obtainable over a network, where the  
 symbolic names are character strings that do not con-  
 tain either a persistent address or pointer to an output  
 value accessible to the web service, and b) an address  
 of the web service, and where each defined UI object is  
 either: 1) selected by a user of an authoring tool; or 2)  
 automatically selected by a system as a preferred UI  
 object corresponding to a symbolic name of the web 30  
 component selected by the user of the authoring tool;  
 selecting the symbolic name from said web component  
 corresponding to the defined UI object, where the  
 selected symbolic name has an associated data format  
 class type corresponding to a subclass of UI objects that  
 support the data format type of the symbolic name, and 40  
 has the preferred UI object;  
 associating the selected symbolic name with the defined  
 UI object; and  
 producing an Application including the selected symbolic  
 name of the defined UI object, where said Application 45  
 is a device-independent code, wherein, when the Appli-  
 cation and Player are provided to the device and  
 executed on the device, and when the user of the device  
 provides one or more input values associated with an  
 input symbolic name to an input of the defined UI 50  
 object,

40

1) the device provides the user provided one or more input  
 values and corresponding input symbolic name to the  
 web service,  
 2) the web service utilizes the input symbolic name and  
 the user provided one or more input values for gener-  
 ating one or more output values having an associated  
 output symbolic name,  
 3) said Player receives the output symbolic name and  
 corresponding one or more output values and provides  
 instructions for the display of the device to present an  
 output value in the defined UI object.

16. The method of claim 15, where said registry includes  
 definitions of input and output related to said web service.

17. The method of claim 15, where said web component  
 is a text chat, a video chat, an image, a slideshow, a video,  
 or an RSS feed.

18. The method of claim 15, where said UI object is an  
 input field for a chat.

19. The method of claim 15, where said UI object is an  
 input field for a web service.

20. The method of claim 15, where said UI object is an  
 input field usable to obtain said web component, where said  
 input field includes a text field, a scrolling text box, a check  
 box, a drop down-menu, a list menu, or a submit button.

21. The method of claim 15, where said web component  
 is an output of a web service, is the text provided by one or  
 more simultaneous chat sessions, is the video of a video chat  
 session, is a video, an image, a slideshow, an RSS display,  
 or an advertisement.

22. The method of claim 15, further comprising:  
 defining a phone field or list; and  
 generating code that, when executed on the device, allows  
 a user to supply a phone number to said phone field or  
 list.

23. The method of claim 15, further comprising:  
 defining a SMS field or list; and  
 generating code that, when executed on the device, allows  
 a user to supply an SMS address to said SMS field or  
 list.

24. The method of claim 15, and such that said Player  
 interprets dynamically received, device independent values  
 of the web component defined in the Application.

25. The method of claim 15, further comprising:  
 providing said Application and Player over said network.

26. The method of claim 15, wherein said UI object  
 corresponds to a widget.

27. The method of claim 15, where said Player is activated  
 and runs in a web browser.

28. The method of claim 15, where said Player is a native  
 program.

\* \* \* \* \*



US009928044B2

(12) **United States Patent**  
**Rempell et al.**

(10) **Patent No.:** **US 9,928,044 B2**

(45) **Date of Patent:** **\*Mar. 27, 2018**

(54) **SYSTEMS AND METHODS FOR  
PROGRAMMING MOBILE DEVICES**

(71) Applicant: **Express Mobile, Inc.**, Novato, CA (US)

(72) Inventors: **Steven H. Rempell**, Novato, CA (US);  
**David Chrobak**, Clayton, CA (US);  
**Ken Brown**, San Martin, CA (US)

(73) Assignee: **Express Mobile, Inc.**, Novato, CA (US)

(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **15/706,746**

(22) Filed: **Sep. 17, 2017**

(65) **Prior Publication Data**

US 2018/0004493 A1 Jan. 4, 2018

**Related U.S. Application Data**

(63) Continuation of application No. 15/370,990, filed on Dec. 6, 2016, now Pat. No. 9,766,864, which is a (Continued)

(51) **Int. Cl.**  
**G06F 3/048** (2013.01)  
**G06F 9/44** (2018.01)  
**H04L 29/08** (2006.01)  
**G06F 3/0484** (2013.01)  
**G06F 3/0482** (2013.01)  
**H04L 12/58** (2006.01)  
**H04L 29/06** (2006.01)

(52) **U.S. Cl.**

CPC ..... **G06F 8/38** (2013.01); **G06F 3/0482** (2013.01); **G06F 3/04842** (2013.01); **G06F 8/34** (2013.01); **G06F 9/4443** (2013.01); **H04L 51/046** (2013.01); **H04L 65/60** (2013.01); **H04L 67/02** (2013.01)

(58) **Field of Classification Search**

CPC ..... G06F 3/048  
See application file for complete search history.

(56) **References Cited**

**U.S. PATENT DOCUMENTS**

2004/0055017 A1 3/2004 Delpuch et al.  
2004/0163020 A1 8/2004 Sidman  
(Continued)

**OTHER PUBLICATIONS**

Stina Nylander et al. "The Ubiquitous Interactor-Device Independent Access to Mobile Services" (Computer-Aided Design for User Interfaces IV, Proceedings of the Fifth International Conference on Computer-Aided Design of User Interfaces CADUI'2004, Jan. 2004, pp. 271-282).\*

(Continued)

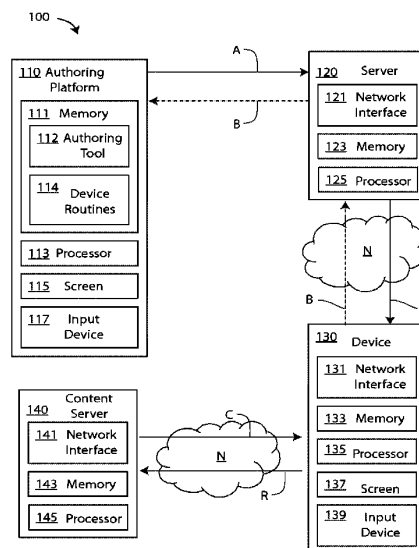
*Primary Examiner* — Xuyang Xia

(74) *Attorney, Agent, or Firm* — Steven R. Vosen

(57) **ABSTRACT**

Embodiments of a system and method are described for generating and distributing programming to mobile devices over a network. Devices are provided with Players specific to each device and Applications that are device independent. Embodiments include a full-featured WYSIWYG authoring environment, including the ability to bind web components to objects.

**28 Claims, 18 Drawing Sheets**



**US 9,928,044 B2**

Page 2

**Related U.S. Application Data**

continuation of application No. 14/708,094, filed on May 8, 2015, now Pat. No. 9,542,163, which is a continuation of application No. 12/936,395, filed as application No. PCT/US2009/039695 on Apr. 6, 2009, now Pat. No. 9,063,755.

- (60) Provisional application No. 61/123,438, filed on Apr. 7, 2008, provisional application No. 61/113,471, filed on Nov. 11, 2008, provisional application No. 61/166,651, filed on Apr. 3, 2009.

- (56) **References Cited**

**U.S. PATENT DOCUMENTS**

2004/0199614 A1\* 10/2004 Shenfield ..... H04L 29/06  
709/220

2005/0149935 A1 7/2005 Benedetti  
2005/0273705 A1\* 12/2005 McCain ..... G06F 17/24  
715/234

2006/0063518 A1 3/2006 Paddon et al.

**OTHER PUBLICATIONS**

International Search Report and Written Opinion—PCT/US2009/039695—Aug. 21, 2009.

International Preliminary Report on Patentability and Written Opinion—PCT/US2009/039695—Oct. 21, 2010.

Rempell et al, co-owned U.S. Pat. No. 9,471,287, Issue date of Oct. 18, 2016.

Rempell et al, co-owned U.S. Pat. No. 9,507,571, Issue date of Nov. 29, 2016.

Rempell et al, co-owned U.S. Pat. No. 9,542,163, Issue date of Nov. 29, 2016.

Rempell et al, co-owned U.S. Pat. No. 9,766,864, Issue date of Sep. 19, 2017.

\* cited by examiner

U.S. Patent

Mar. 27, 2018

Sheet 1 of 18

US 9,928,044 B2

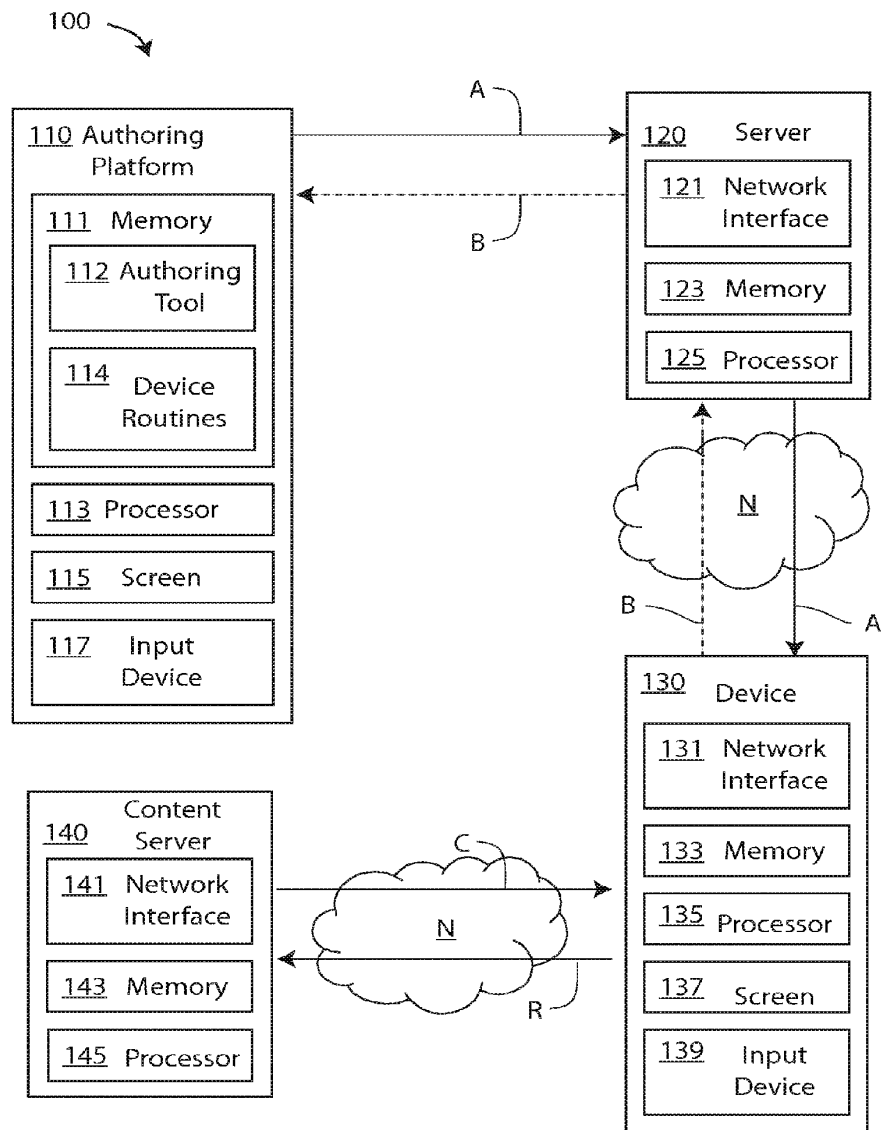


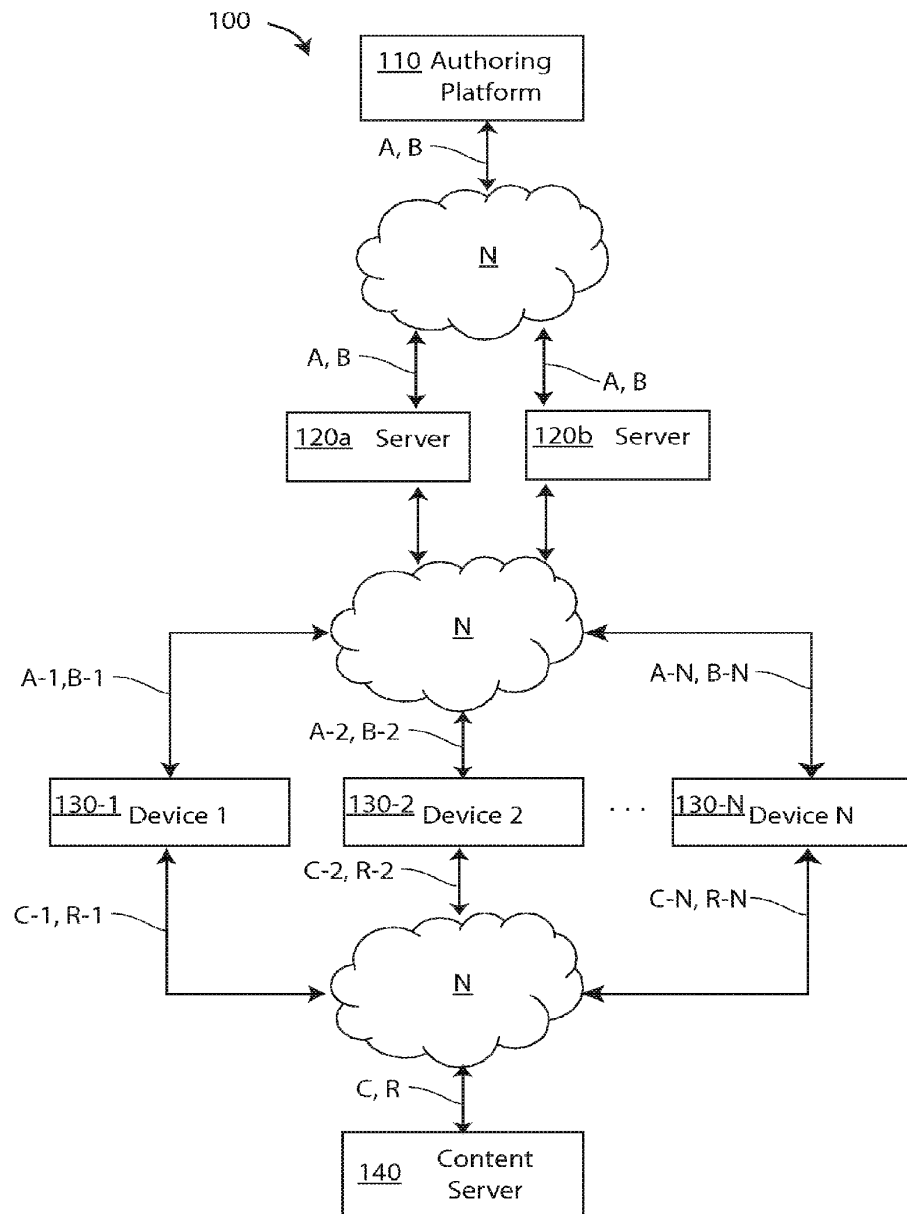
FIG. 1A

**U.S. Patent**

**Mar. 27, 2018**

**Sheet 2 of 18**

**US 9,928,044 B2**



**FIG. 1B**

U.S. Patent

Mar. 27, 2018

Sheet 3 of 18

US 9,928,044 B2

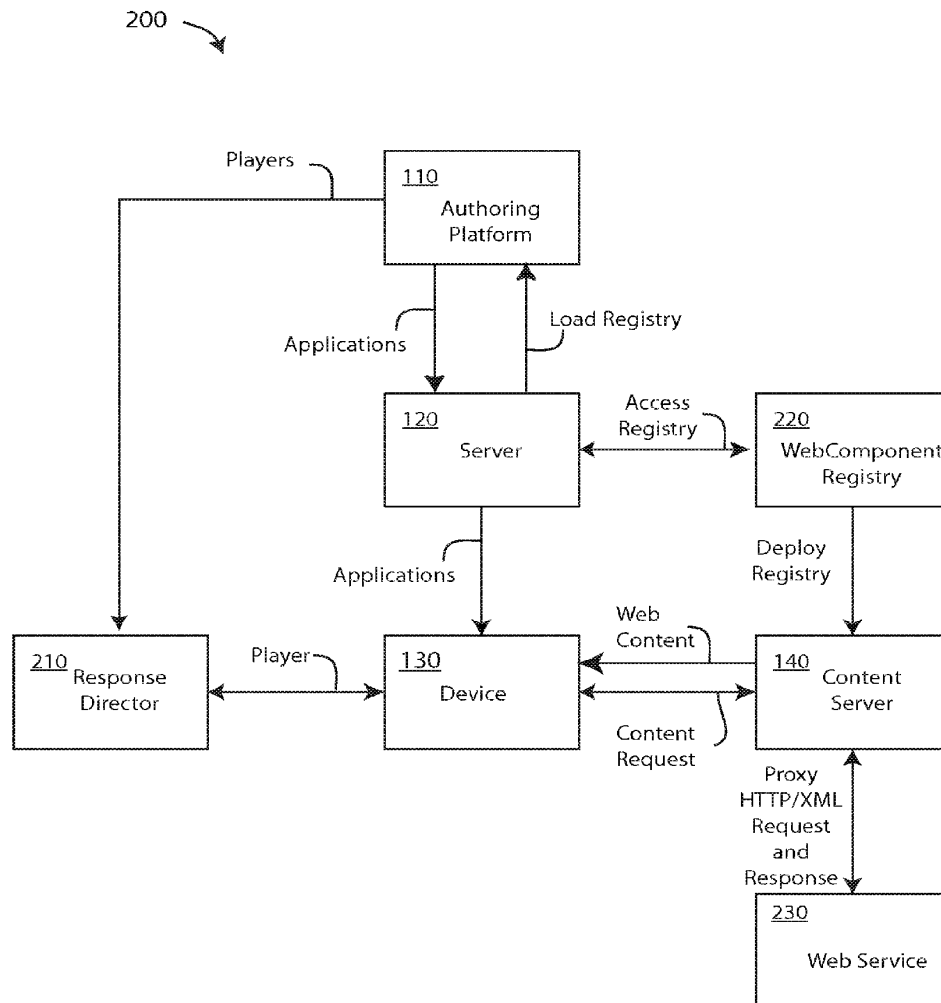


FIG. 2A



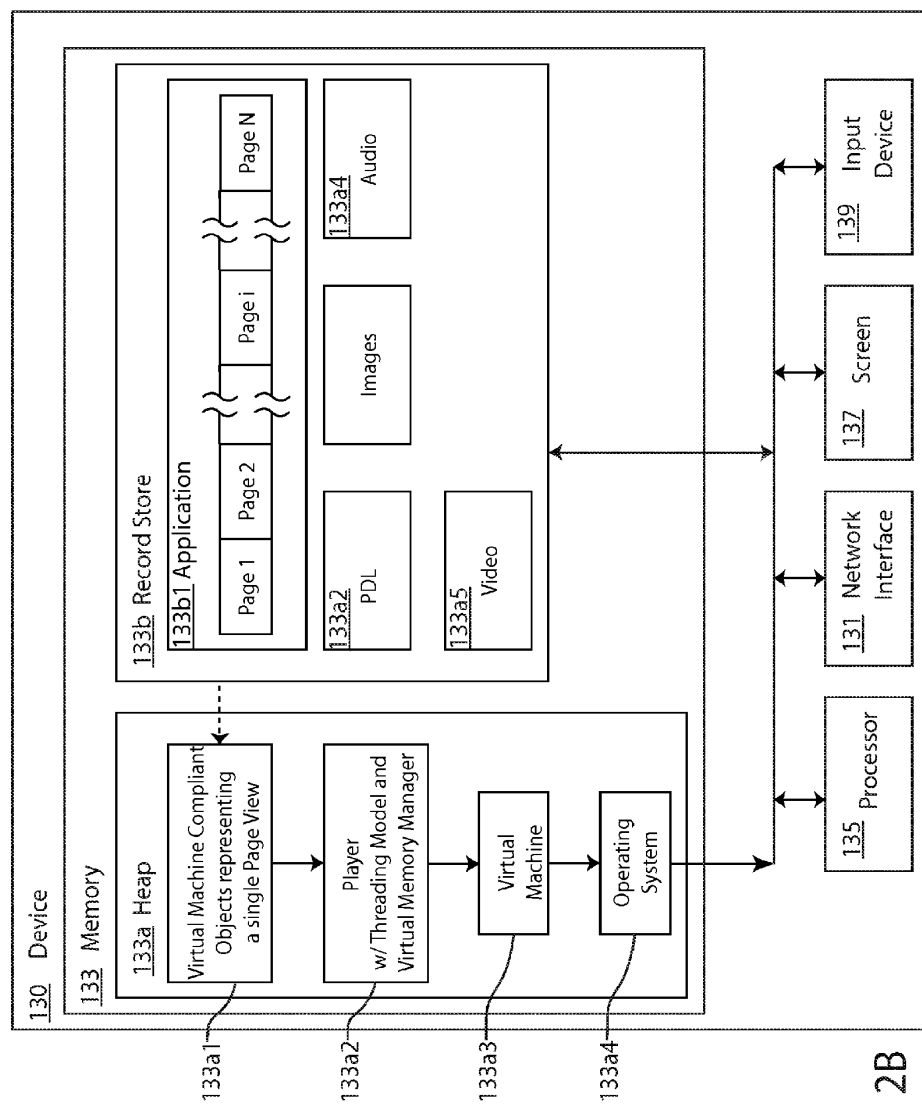


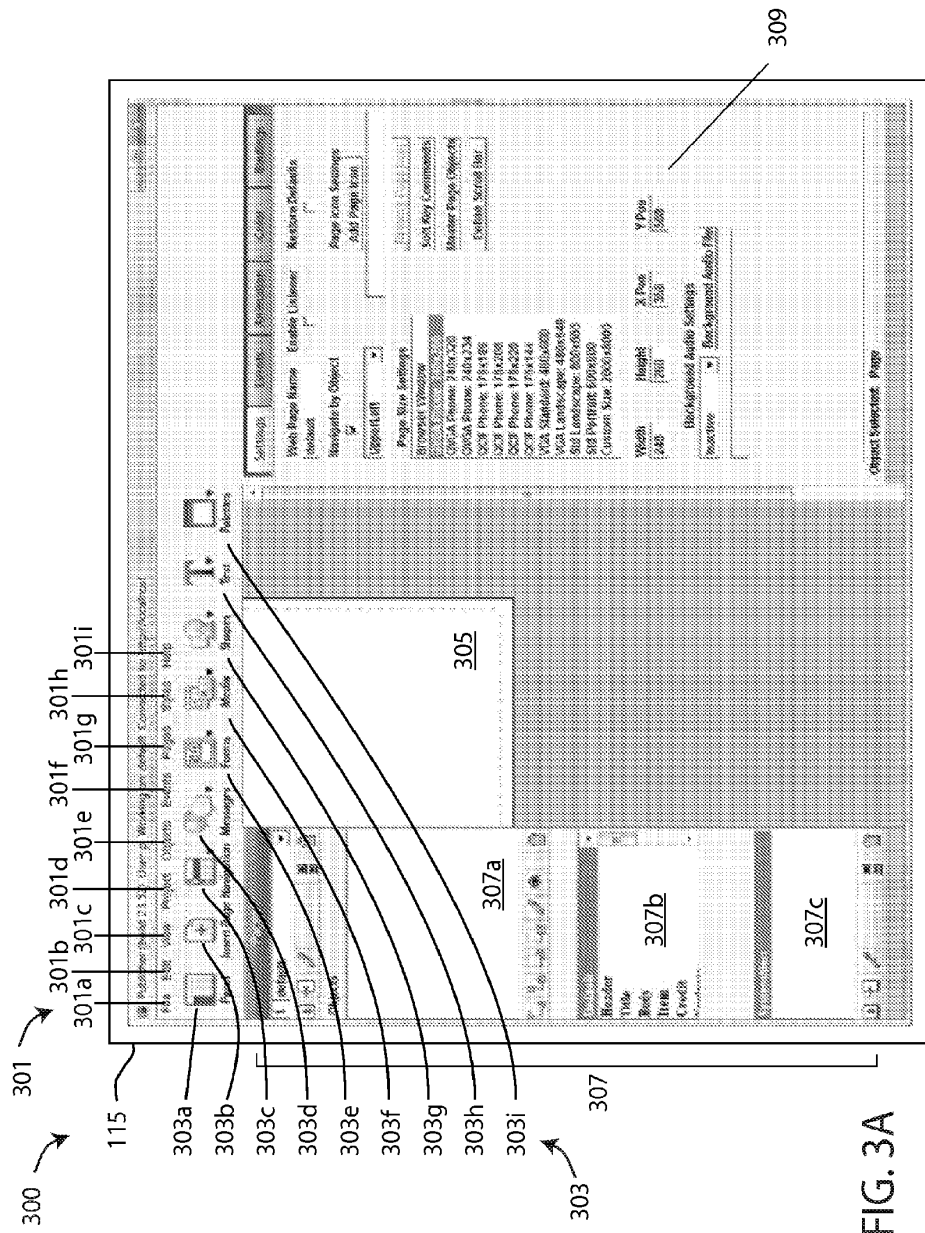
FIG. 2B

U.S. Patent

Mar. 27, 2018

Sheet 5 of 18

US 9,928,044 B2

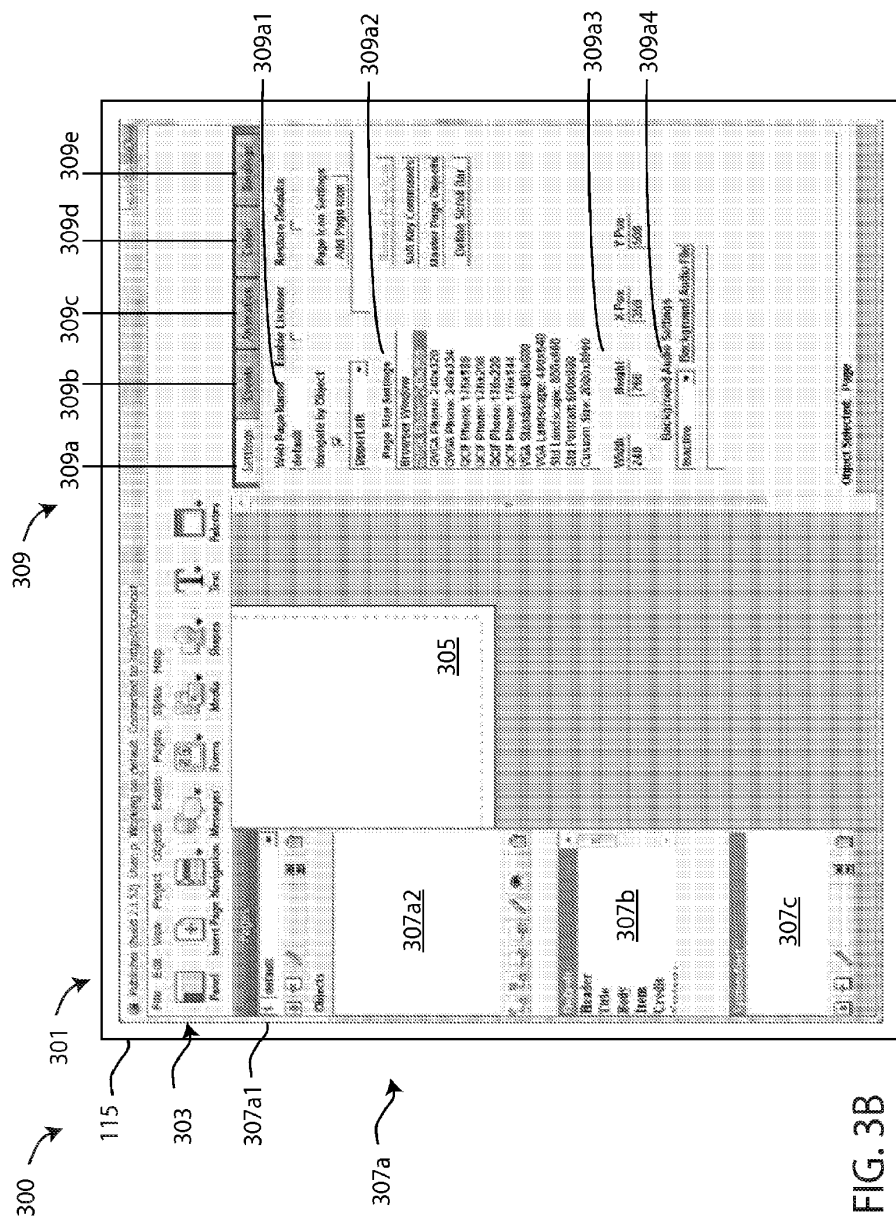


U.S. Patent

Mar. 27, 2018

Sheet 6 of 18

US 9,928,044 B2



U.S. Patent

Mar. 27, 2018

Sheet 7 of 18

US 9,928,044 B2

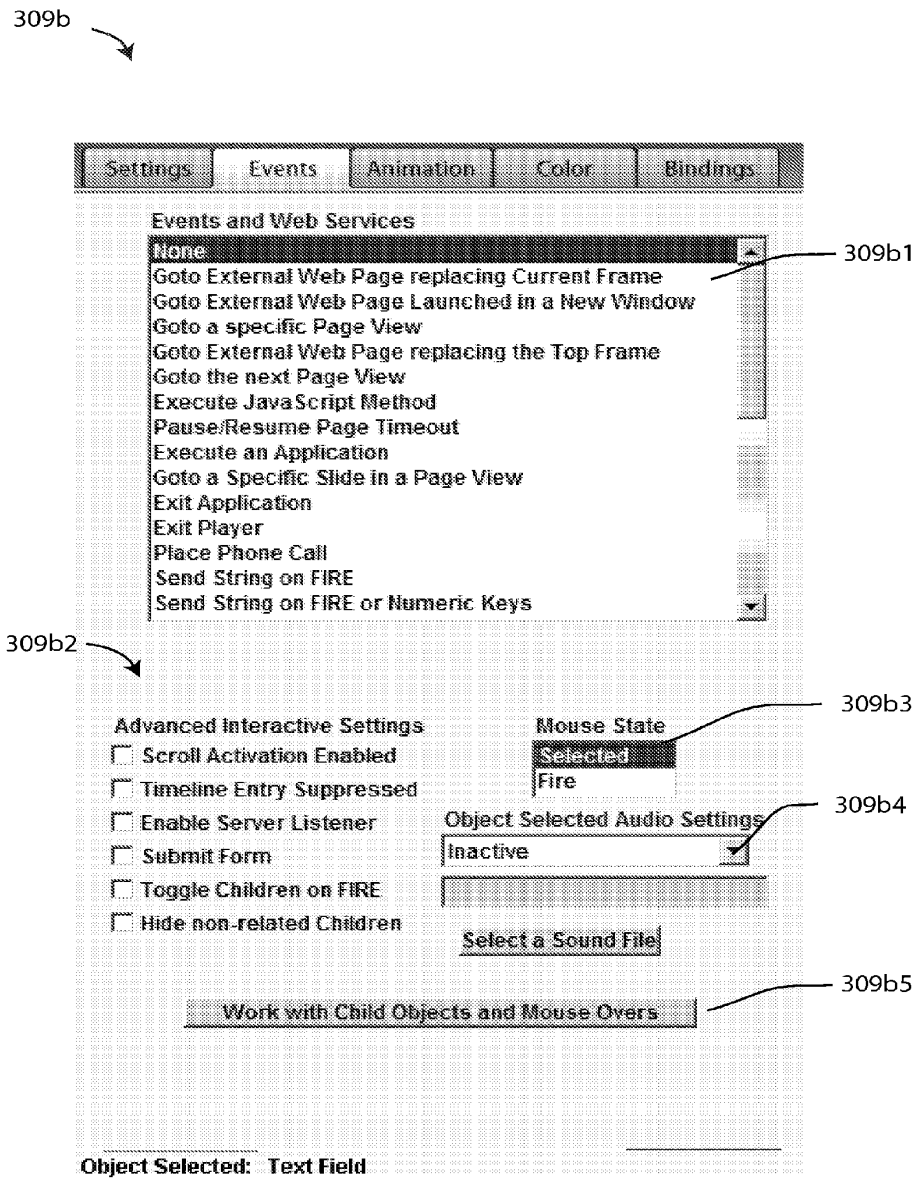


FIG. 3C

U.S. Patent

Mar. 27, 2018

Sheet 8 of 18

US 9,928,044 B2

309c

Settings Events Animation Color Bindings

Object Entry Timeline Specifications

Activate Timeline Delay (Sec) Direction Movement Duration(Sec) Frames

☐ 0 0 None None 2 0 10

Specifications for this Object's Entry Animation Audio Track

Inactive Entry Audio File

Object Animation Specifications

Delay(Sec)	Direction	Movement	Duration(Sec)	Frames
0	None	None	0	1
1	Scroll Left	Fade	1	2
2	Scroll Right	Fade In	2	3
3	Custom	Fade Out	3	4
4	Multi-Point		4	5
5	Seek Cursor		5	6
6	Attach		6	7
7	Deposit		7	8
8	Send Home		8	9
9	Carom N		9	10
10	Carom NE		10	11

Pathname for this Object's Animation Audio Track

Inactive Main Audio File

Animation Cycles Custom Zoom % Avoid Cursor Dampen Anim.

1 0 ☒ ☐

Object Exit Timeline Specifications

Activate Timeline Delay (Sec) Direction Movement Duration(Sec) Frames

☐ 0 0 None None 2 0 10

Specifications for this Object's Exit Animation Audio Track

Inactive Exit Audio File

FIG. 3D

U.S. Patent

Mar. 27, 2018

Sheet 9 of 18

US 9,928,044 B2

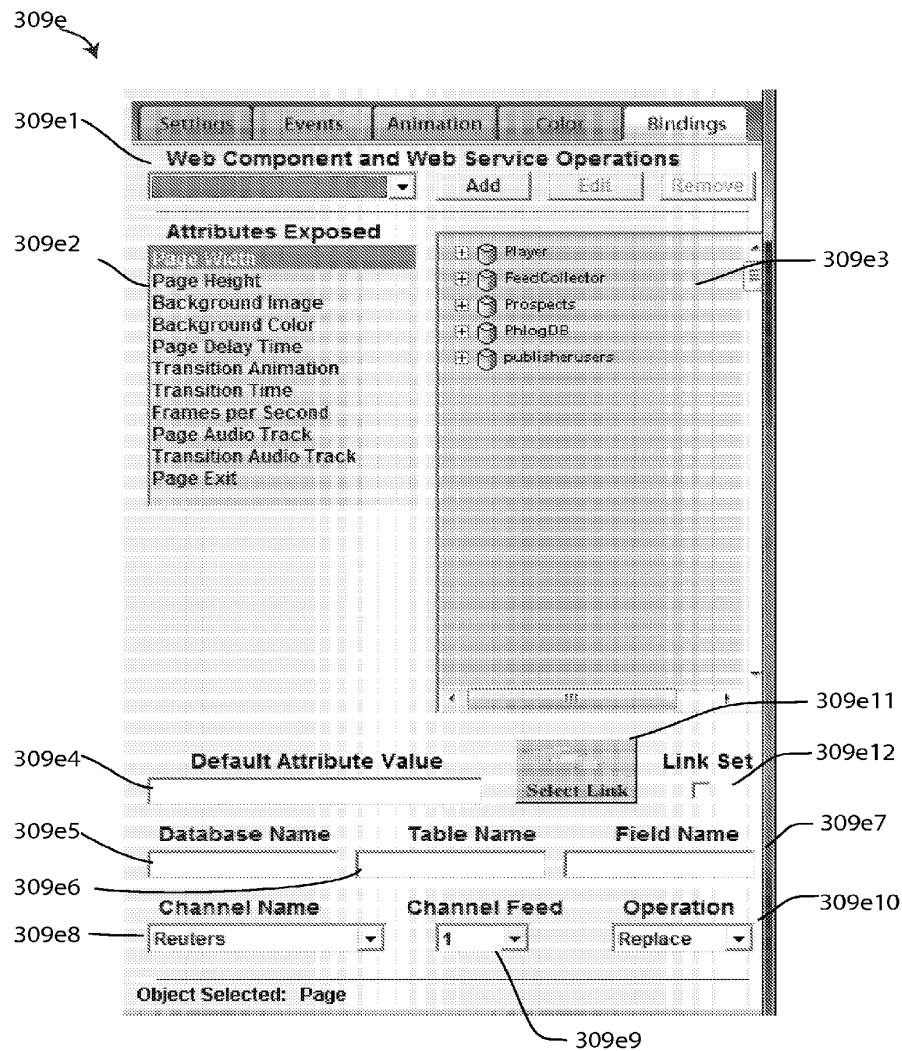


FIG. 3E

U.S. Patent

Mar. 27, 2018

Sheet 10 of 18

US 9,928,044 B2

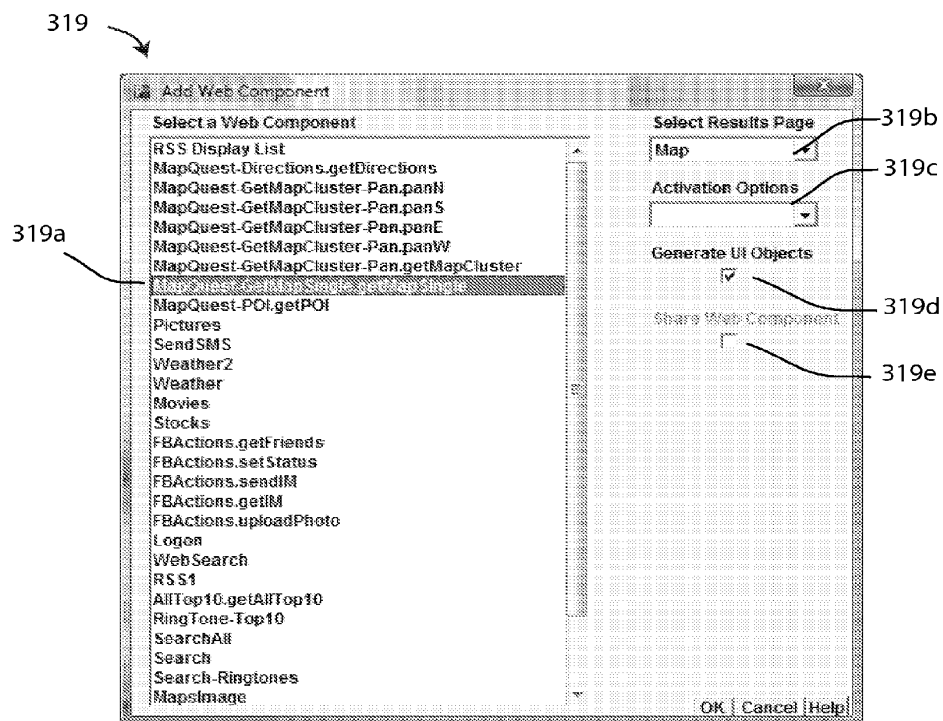


FIG. 3F

U.S. Patent

Mar. 27, 2018

Sheet 11 of 18

US 9,928,044 B2

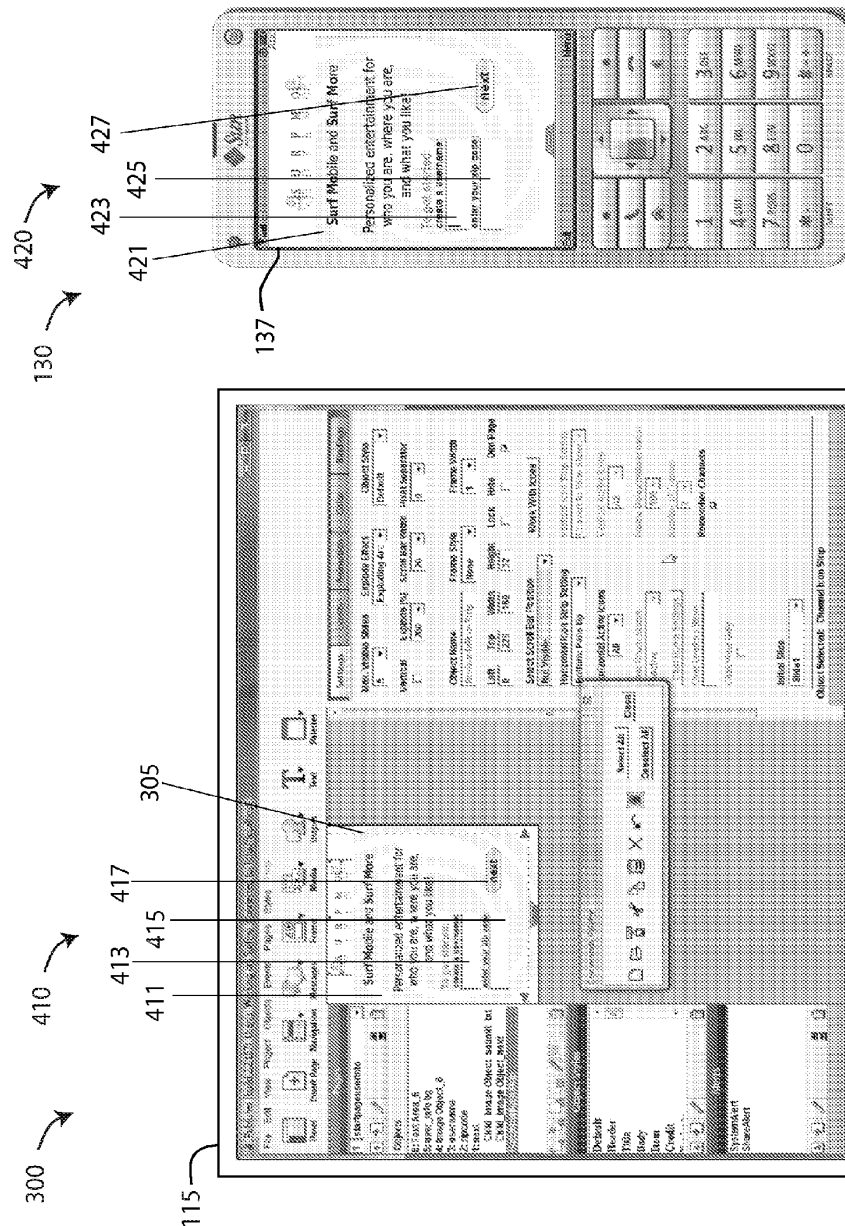


FIG. 4B

FIG. 4A



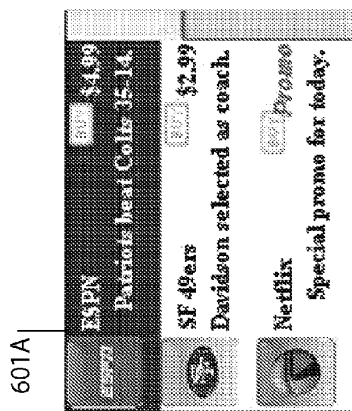


FIG. 6A

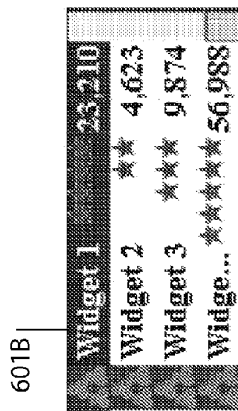


FIG. 6B

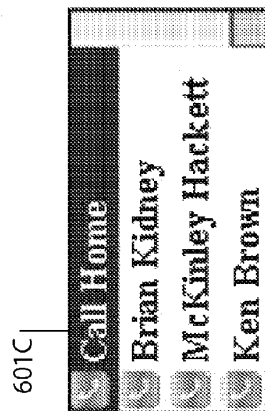


FIG. 6C

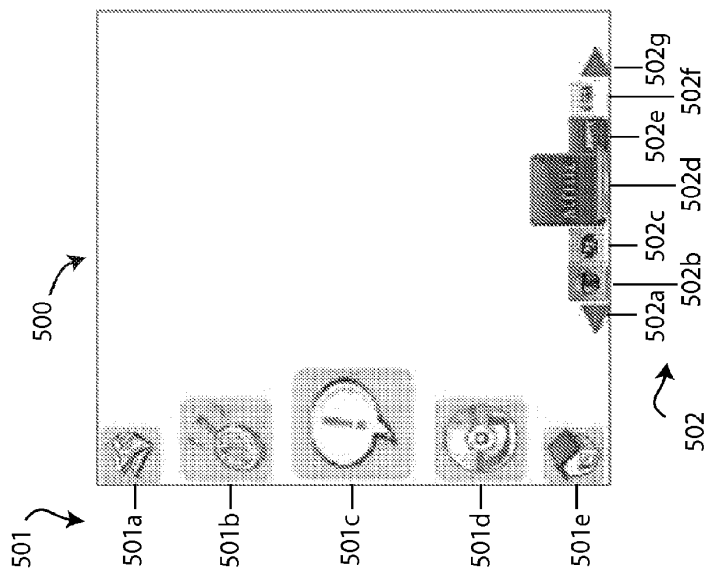


FIG. 5

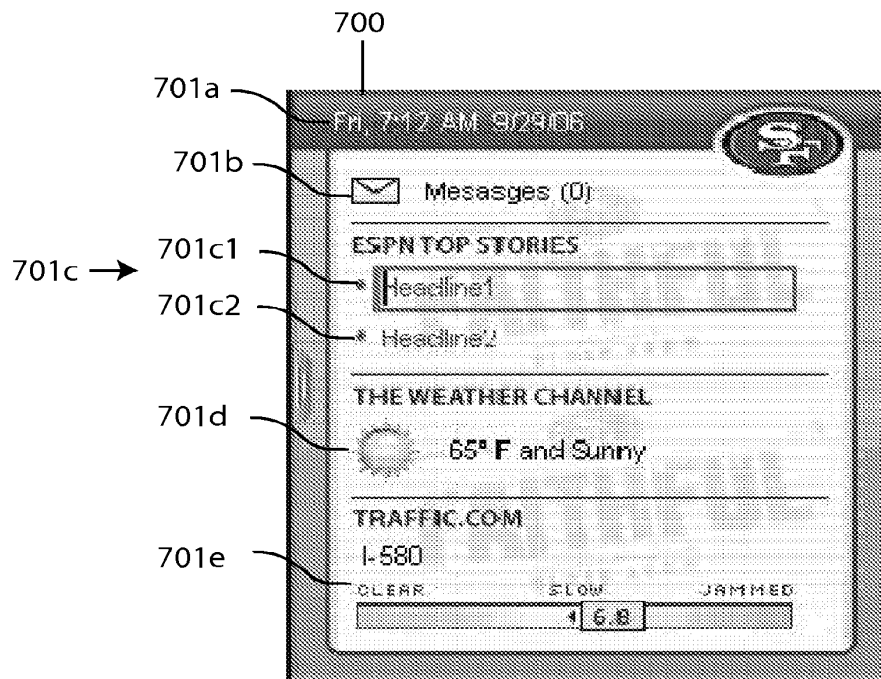


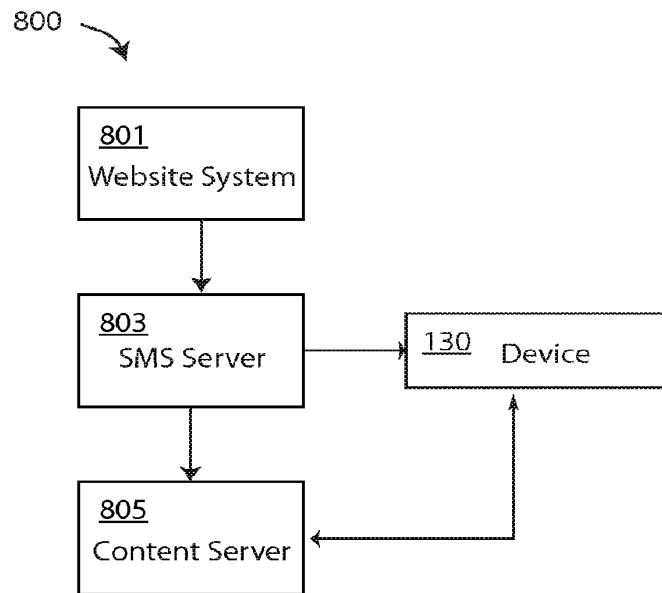
FIG. 7

**U.S. Patent**

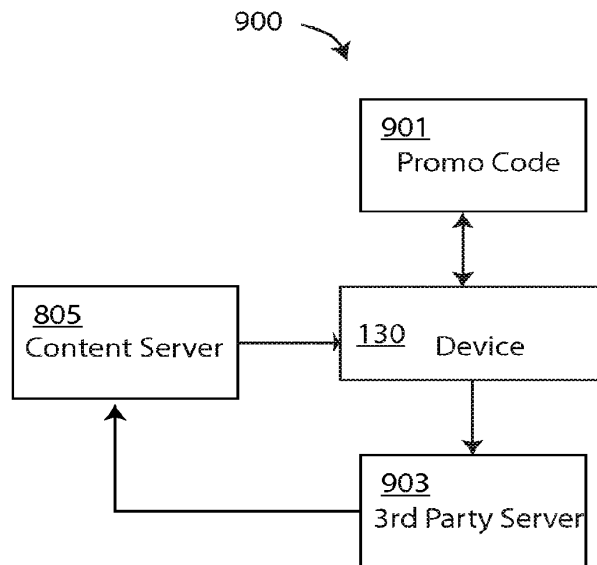
**Mar. 27, 2018**

**Sheet 14 of 18**

**US 9,928,044 B2**



**FIG. 8**



**FIG. 9**

U.S. Patent

Mar. 27, 2018

Sheet 15 of 18

US 9,928,044 B2

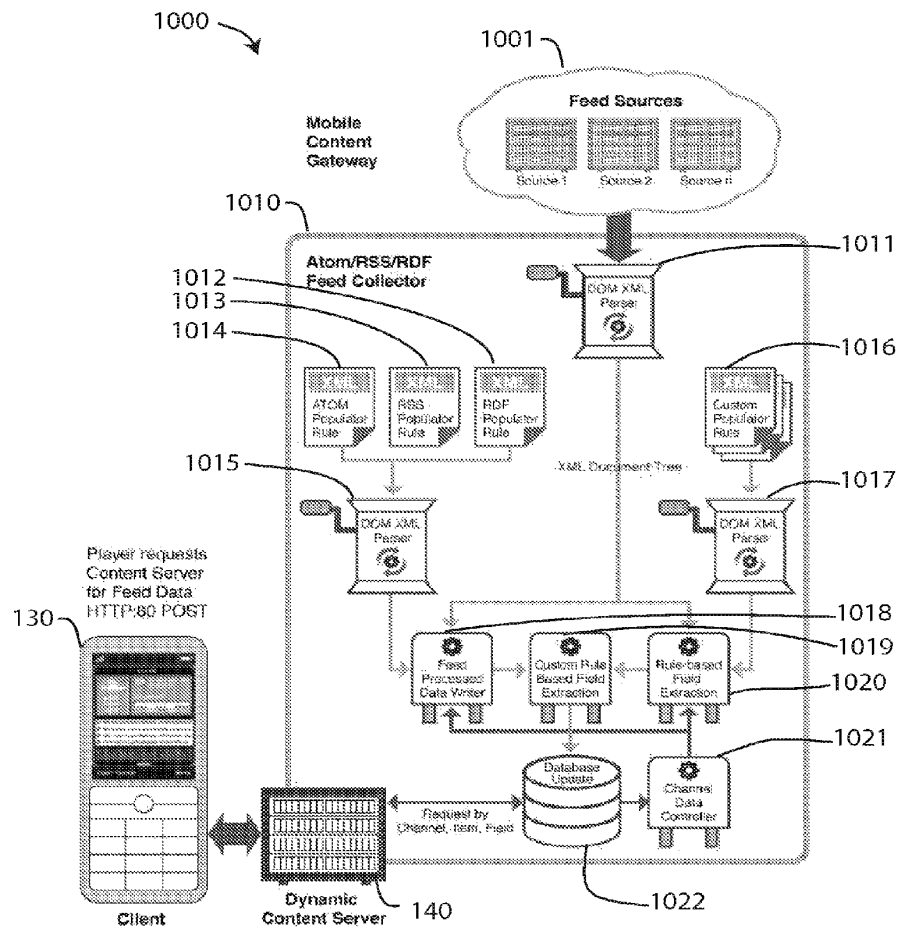


FIG. 10

U.S. Patent

Mar. 27, 2018

Sheet 16 of 18

US 9,928,044 B2

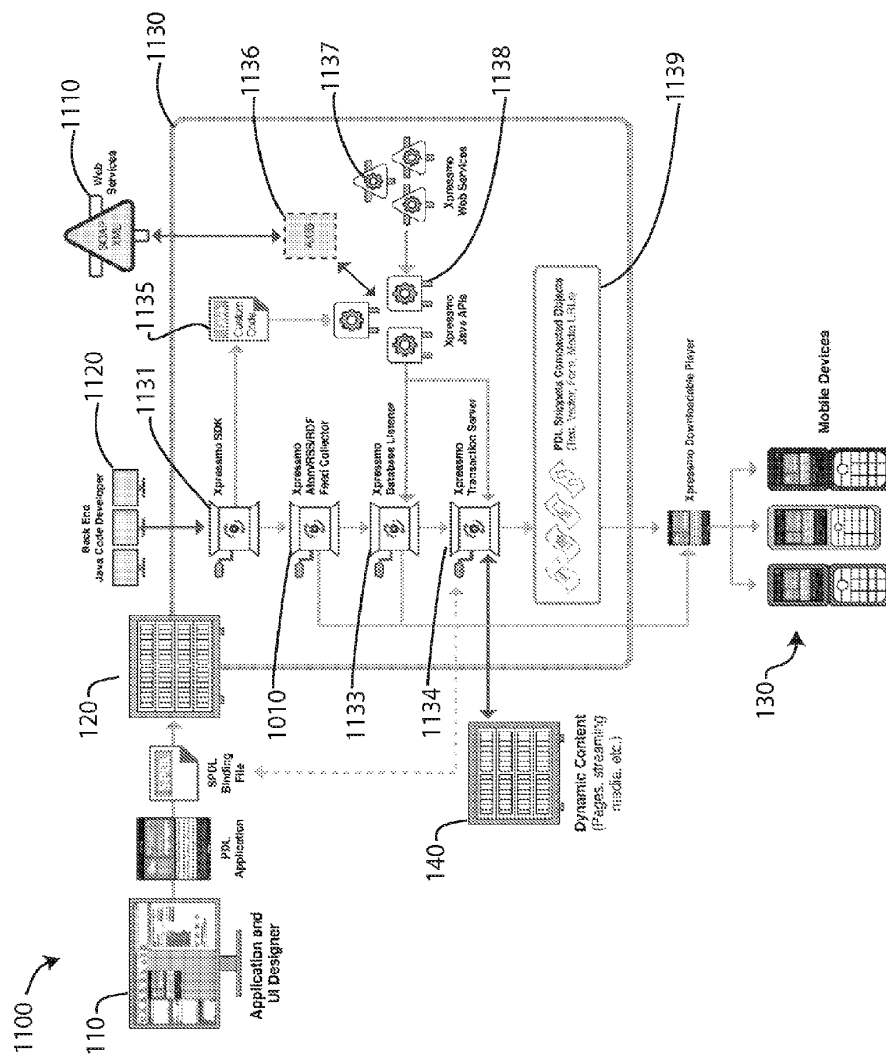


FIG. 11

U.S. Patent

Mar. 27, 2018

Sheet 17 of 18

US 9,928,044 B2

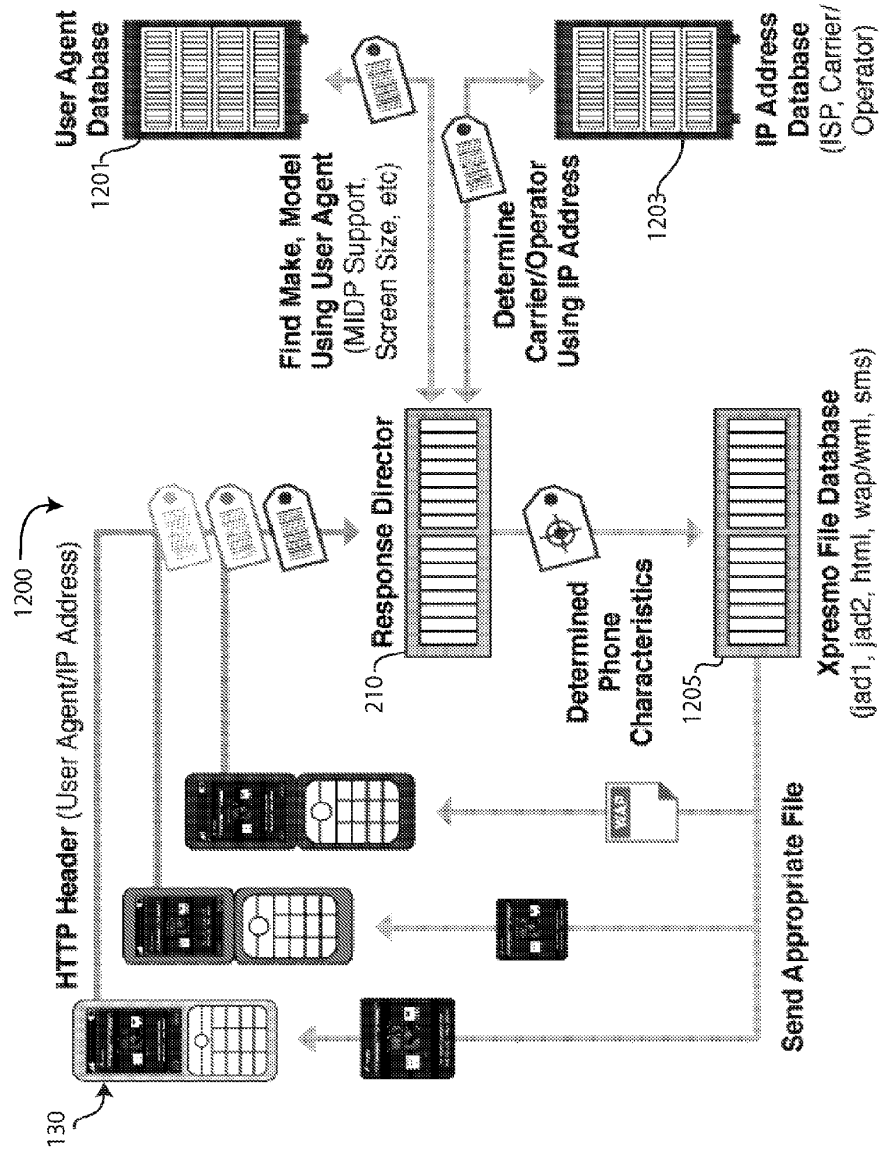


FIG. 12

U.S. Patent

Mar. 27, 2018

Sheet 18 of 18

US 9,928,044 B2

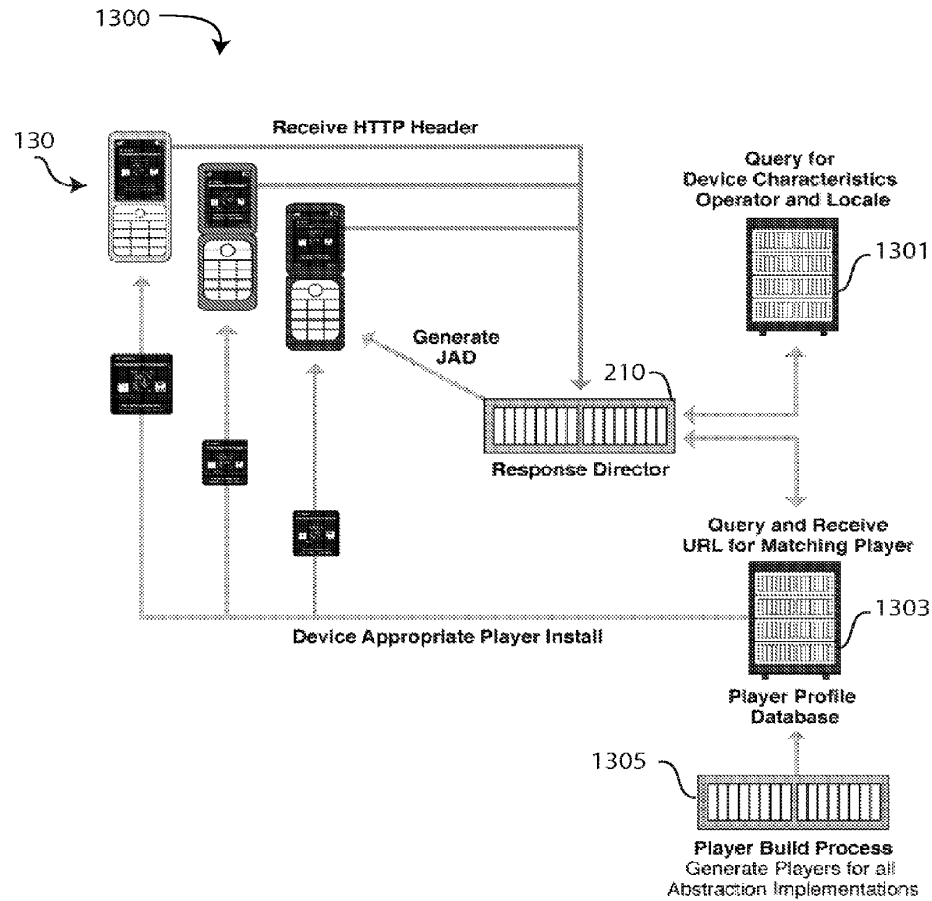


FIG. 13

US 9,928,044 B2

1

**SYSTEMS AND METHODS FOR  
PROGRAMMING MOBILE DEVICES****TECHNICAL FIELD**

The present invention generally relates to providing software for mobile devices, and more particularly to a method and system for authoring Applications for devices.

**BACKGROUND ART**

Internet-connected mobile devices are becoming ever more popular. While these devices provide portability to the Internet, they generally do not have the capabilities of non-mobile devices including computing, input and output capabilities.

In addition, the mobility of the user while using such devices provides challenges and opportunities for the use of the Internet. Further, unlike non-mobile devices, there are a large number of types of devices and they tend to have a shorter lifetime in the marketplace. The programming of the myriad of mobile devices is a time-consuming and expensive proposition, thus limiting the ability of service providers to update the capabilities of mobile devices.

Thus there is a need in the art for a method and apparatus that permits for the efficient programming of mobile devices. Such a method and apparatus should be easy to use and provide output for a variety of devices.

**DISCLOSURE OF INVENTION**

In certain embodiments, a system is provided to generate code to provide content on a display of a platform. The system includes a database of web services obtainable over a network and an authoring tool. The authoring tool is configured to define an object for presentation on the display, select a component of a web service included in said database, associate said object with said selected component, and produce code that, when executed on the platform, provides said selected component on the display of the platform.

In certain other embodiments, a method is provided for providing information to platforms on a network. The method includes accepting a first code over the network, where said first code is platform-dependent; providing a second code over the network, where said second code is platform-independent; and executing said first code and said second code on the platform to provide web components obtained over the network.

In certain embodiments, a method for displaying content on a platform utilizing a database of web services obtainable over a network is provided. The method includes: defining an object for presentation on the display; selecting a component of a web service included in said database; associating said object with said selected component; and producing code that, when executed on the platform, provides said selected component on the display of the platform.

In one embodiment, one of the codes is a Player, which is a thin client architecture that operates in a language that manages resources efficiently, is extensible, supports a robust application model, and has no device specific dependencies. In another embodiment, Player P is light weight and extends the operating system and/or virtual machine of the device to: Manage all applications and application upgrades, and resolve device, operating system, VM and language fragmentation.

2

In another embodiment, one of the codes is an Application that is a device independent code that interpreted by the Player.

These features together with the various ancillary provisions and features which will become apparent to those skilled in the art from the following detailed description, are attained by the system and method of the present invention, preferred embodiments thereof being shown with reference to the accompanying drawings, by way of example only, wherein:

**BRIEF DESCRIPTION OF DRAWINGS**

FIG. 1A is an illustrative schematic of one embodiment of a system including an authoring platform and a server for providing programming instructions to a device over a network;

FIG. 1B is schematic of an alternative embodiment system for providing programming instructions to device over a network;

FIG. 2A is a schematic of an embodiment of system illustrating the communications between different system components;

FIG. 2B is a schematic of one embodiment of a device illustrating an embodiment of the programming generated by authoring platform;

FIGS. 3A and 3B illustrate one embodiment of a publisher interface as it appears, for example and without limitation, on a screen while executing an authoring tool;

FIG. 3C illustrates an embodiment of the Events Tab;

FIG. 3D illustrates one embodiment of an Animation Tab;

FIG. 3E illustrates one embodiment of Bindings Tab;

FIG. 3F illustrates one embodiment of a pop-up menu for adding web components;

FIG. 4A shows a publisher interface having a layout on a canvas; and FIG. 4B shows a device having the resulting layout on a device screen;

FIG. 5 shows a display of launch strips;

FIG. 6A is a display of a Channel Selection List;

FIG. 6B is a display of a Widget Selection List;

FIG. 6C is a display of a Phone List;

FIG. 7 shows a display of a mash-up;

FIG. 8 is a schematic of an embodiment of a push capable system;

FIG. 9 is a schematic of an alternative embodiment of a push capable system;

FIG. 10 is a schematic of one embodiment of a feed collector;

FIG. 11 is a schematic of an embodiment of a Mobile Content Gateway;

FIG. 12 is a schematic of one embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database; and

FIG. 13 is a schematic of another embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database.

Reference symbols are used in the Figures to indicate certain components, aspects or features shown therein, with reference symbols common to more than one Figure indicating like components, aspects or features shown therein.

**MODE(S) FOR CARRYING OUT THE  
INVENTION**

FIG. 1A is an illustrative schematic of one embodiment of a system **100** including an authoring platform **110** and a server **120** for providing programming instructions to a



US 9,928,044 B2

3

device 130 over a network N. In one embodiment, device 130 is a wireless device, and network N includes wireless communication to the device. Alternatively, system 100 may provide access over network N to other information, data, or content, such as obtainable as a web service over the Internet. In general, a user of authoring platform 110 may produce programming instructions or files that may be transmitted over network N to operate device 130, including instructions or files that are sent to device 130 and/or server 120. The result of the authoring process is also referred to herein, and without limitation, as publishing an Application.

Embodiments include one or more databases that store information related to one or more devices 130 and/or the content provided to the devices. It is understood that such databases may reside on any computer or computer system on network N, and that, in particular, the location is not limited to any particular server, for example.

Device 130 may be, for example and without limitation, a cellular telephone or a portable digital assistant, includes a network interface 131, a memory 133, a processor 135, a screen 137, and an input device 139. Network interface 131 is used by device 130 to communication over a wireless network, such as a cellular telephone network, a WiFi network or a WiMax network, and then to other telephones through a public switched telephone network (PSTN) or to a satellite, or over the Internet. Memory 133 includes programming required to operate device 130 (such as an operating system or virtual machine instructions), and may include portions that store information or programming instructions obtained over network interface 131, or that are input by the user (such as telephone numbers or images from a device camera (not shown)). In one embodiment screen 137 is a touch screen, providing the functions of the screen and input device 139.

Authoring platform 110 includes a computer or computer system having a memory 111, a processor 113, a screen 115, and an input device 117. It is to be understood that memory 111, processor 113, screen 115, and input device 117 are configured such a program stored in the memory may be executed by the processor to accept input from the input device and display information on the screen. Further, the program stored in memory 111 may also instruct authoring platform 110 to provide programming or information, as indicated by the line labeled "A" and to receive information, as indicated by the line labeled "B."

Memory 111 is shown schematically as including a stored program referred to herein, and without limitation, as an authoring tool 112. In one embodiment, authoring tool 112 is a graphical system for designing the layout of features as a display that is to appear on screen 137. One example of authoring tool 112 is the CDERT<sup>TM</sup> publishing platform (Express Mobile, Inc., Novato, Calif.).

In another embodiment, which is not meant to limit the scope of the present invention, device 130 may include an operating system having a platform that can interpret certain routines. Memory 111 may optionally include programming referred to herein, and without limitation, as routines 114 that are executable on device 130.

Routines 114 may include device-specific routines—that is, codes that are specific to the operating system, programming language, or platform of specific devices 130, and may include, but are not limited to, Java, Windows Mobile, Brew, Symbian OS, or Open Handset Alliance (OHA). Several examples and embodiments herein are described with reference to the use of Java. It is to be understood that the invention is not so limited, except as provided in the claims, and that one skilled in the art could provide Players for

4

devices using routines provided on a platform. Thus as an example, routines 114 may include Java API's and an authoring tool System Development Kit (SDK) for specific devices 130.

Server 120 is a computer or computer system that includes a network interface 121, a memory 123, and a processor 125. Is to be understood that network interface 121, memory 123, and processor 125 are configured such that a program stored in the memory may be executed by the processor to: accept input and/or provide output to authoring platform 110; accept input and/or provide output through network interface 121 over network N to network interface 131; or store information from authoring platform 110 or from device 130 for transmission to another device or system at a later time.

In one embodiment, authoring platform 110 permits a user to design desired displays for screen 137 and actions of device 130. In other words, authoring platform 110 is used to program the operation of device 130. In another embodiment, authoring platform 110 allows a user to provide input for the design of one or more device displays and may further allow the user to save the designs as device specific Applications. The Applications may be stored in memory 123 and may then be sent, when requested by device 130 or when the device is otherwise accessible, over network N, through network interface 130 for storage in memory 133.

In an alternative embodiment, analytics information from devices 130 may be returned from device 130, through network N and server 120, back to authoring platform 110, as indicated by line B, for later analysis. Analytics information includes, but is not limited to, user demographics, time of day, and location. The type of analytic content is only limited by which listeners have been activated for which objects and for which pages. Analytic content may include, but is not limited to, player-side page view, player-side forms-based content, player-side user interactions, and player-side object status.

Content server 140 is a computer or computer system that includes a network interface 141, a memory 143, and a processor 145. It is to be understood that network interface 141, memory 143, and processor 145 are configured such that a stored program in the memory may be executed by the processor to accepts requests R from device 130 and provide content C over a network, such as web server content the Internet, to device 130.

FIG. 1B is schematic of an alternative embodiment system 100 for providing programming instructions to device 130 over a network N that is generally similar to the system of FIG. 1A. The embodiment of FIG. 1B illustrates that system 100 may include multiple servers 120 and/or multiple devices 130.

In the embodiment of FIG. 1B, system 100 is shown as including two or more servers 120, shown illustratively and without limitation as servers 120a and 120b. Thus some of the programming or information between authoring platform 110 and one or more devices 130 may be stored, routed, updated, or controlled by more than one server 120. In particular, the systems and methods described herein may be executed on one or more server 120.

Also shown in FIG. 1B are a plurality of devices 130, shown illustratively and without limitation as device 130-1, 130-1, . . . 130-N. System 100 may thus direct communication between individual server(s) 120 and specific device(s) 130.

As described subsequently, individual devices 130 may be provided with program instructions which may be stored in each device's memory 133 and where the instructions are

US 9,928,044 B2

5

executed by each device's processor 135. Thus, for example, server(s) 120 may provide device(s) 130 with programming in response to the input of the uses of the individual devices. Further, different devices 130 may be operable using different sets of instructions, that is having one of a variety of different "device platforms." Differing device platforms may result, for example and without limitation, to different operating systems, different versions of an operating system, or different versions of virtual machines on the same operating system. In some embodiments, devices 130 are provided with some programming from authoring system 100 that is particular to the device.

In one embodiment, system 100 provides permits a user of authoring platform 110 to provide instructions to each of the plurality of devices 130 in the form of a device- or device-platform specific instructions for processor 135 of the device, referred to herein and without limitation as a "Player," and a device-independent program, referred to herein and without limitation as an "Application." Thus, for example, authoring platform 110 may be used to generate programming for a plurality of devices 130 having one of several different device platforms. The programming is parsed into instructions used by different device platforms and instructions that are independent of device platform. Thus in one embodiment, device 130 utilizes a Player and an Application to execute programming from authoring platform 110. A device having the correct Player is then able to interpret and be programmed according to the Application.

In one alternative embodiment, the Player is executed the first time by device 130 ("activated") through an Application directory. In another alternative embodiment, the Player is activated by a web browser or other software on device 130. In yet another alternative embodiment, Player is activated through a signal to device 130 by a special telephone numbers, such as a short code.

When the Application and the Player are provided to memory 133, the functioning of device 130 may occur in accordance with the desired programming. Thus in one embodiment, the Application and Player includes programming instructions which may be stored in memory 133 and which, when executed by processor 135, generate the designed displays on screen 137. The Application and Player may also include programming instructions which may be stored in memory 133 and which provide instructions to processor 135 to accept input from input device 139.

Authoring tool 112 may, for example, produce and store within memory 111 a plurality of Players (for different devices 130) and a plurality of Applications for displaying pages on all devices. The Players and Applications are then stored on one or more servers 120 and then provided to individual devices 130. In general, Applications are provided to device 130 for each page of display or a some number of pages. A Player need be provided once or updated as necessary, and thus may be used to display a large number of Applications. This is advantageous for the authoring process, since all of the device-dependent programming is provided to a device only once (or possibly for some small number of upgrades), permitting a smaller Application, which is the same for each device 130.

Thus, for example and without limitation, in one embodiment, the Player transforms device-independent instructions of the Application into device-specific instructions that are executable by device 130. Thus, by way of example and without limitation, the Application may include Java programming for generating a display on screen 137, and the Player may interpret the Java and instruct processor 135 to produce the display according to the Application for execu-

6

tion on a specific device 130 according to the device platform. The Application may in general include, without limitation, instructions for generating a display on screen 137, instructions for accepting input from input device 139, instructions for interacting with a user of device 130, and/or instructions for otherwise operating the device, such as to place a telephone call.

The Application is preferably code in a device-independent format, referred to herein and without limitation as a Portable Description Language (PDL). The device's Player interprets or executes the Application to generate one or more "pages" ("Applications Pages") on a display as defined by the PDL. The Player may include code that is device-specific—that it, each device is provided with a Player that is used in the interpretation and execution of Applications. Authoring tool 112 may thus be used to design one or more device-independent Applications and may also include information on one or more different devices 130 that can be used to generate a Player that specific devices may use to generate displays from the Application.

In one embodiment, system 100 provides Players and Applications to one server 120, as in FIG. 1A. In another embodiment, system 100 provides Players to a first server 120a and Applications to a second server 120b, as in FIG. 1B.

In one embodiment, authoring tool 112 may be used to program a plurality of different devices 130, and routines 114 may include device-specific routines. In another embodiment, the Player is of the type that is commonly referred to as a "thin client"—that is, software for running on the device as a client in client-server architecture with a device network which depends primarily on a central server for processing activities, and mainly focuses on conveying input and output between the user and the server.

In one embodiment, authoring platform 110 allows user to arrange objects for display on screen. A graphical user interface ("GUI," or "UI") is particularly well suited to arranging objects, but is not necessary. The objects may correspond to one or more of an input object, an output object, an action object, or may be a decorative display, such as a logo, or background color or pattern, such as a solid or gradient fill. In another embodiment, authoring platform 110 also permits a user to assign actions to one or more of an input object, an output object, or an action object. In yet another embodiment, authoring platform 110 also permits a user to bind one or more of an input object, an output object, or an action object with web services or web components, or permits a user to provide instructions to processor 135 to store or modify information in memory 133, to navigate to another display or service, or to perform other actions, such as dialing a telephone number.

In certain embodiments, the applicant model used in developing and providing Applications is a PDL. The PDL can be conceptually viewed as a device, operating system and virtual machine agnostic representation of Java serialized objects. In certain embodiments, the PDL is the common language for authoring tool 112, the Application, and Player. Thus while either designing the Application with the authoring tool 112, or programming with the SDK, the internal representation of the programming logic is in Java. In one embodiment the SDK is used within a multi-language software development platform comprising an IDE and a plug-in system to extend it, such as the Eclipse Integrated Development Environment (see, for example, <http://www.eclipse.org/>). At publish time the Java code is translated into

US 9,928,044 B2

7

a PDL. This translation may also occur in real-time during the execution of any Web Services or backend business logic that interacts with the user.

One embodiment for compacting data that may be used is described in co-pending U.S. Pat. No. 6,546,397 to Rempell ("Rempell"), the contents of which are incorporated herein by reference. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

The use of a PDL, as described in Rempell, permits for efficient code and data compaction. Code, as well as vector, integer and Boolean data may be compacted and then compressed resulting in a size reduction of 40 to 80 times that of the original Java serialized objects. This is important not only for performance over the network but for utilizing the virtual memory manager of the Player more efficiently. As an example, the reassembled primitives of the Java objects may first undergo logical compression, followed by LZ encoding.

The use of a PDL also provides virtual machine and operating system independence. Since the reassembled primitives of the Application no longer have any dependencies from the original programming language (Java) that they were defined in. The PDL architecture takes full advantage of this by abstracting all the virtual machine and/or operating system interfaces from the code that processes the PDL.

In one embodiment, the PDL is defined by the means of nested arrays of primitives. Accordingly, the use of a PDL provides extensibility and compatibility, with a minimal amount of constraints in extending the Player seamlessly as market demands and device capabilities continue to grow. Compatibility with other languages is inherent based on the various Player abstraction implementations, which may be, for example and without limitation, Java CDC, J2SE or MIDP2 implementations.

In one embodiment, the architecture of Player P includes an abstraction interface that separates all device, operating system and virtual machine dependencies from the Player's Application model business logic (that is, the logic of the server-side facilities) that extend the Application on the Player so that it is efficiently integrated into a comprehensive client/server Application. The use of an abstraction interface permits the more efficient porting to other operating systems and virtual machines and adding of extensions to the Application model so that a PDL can be implemented once and then seamlessly propagated across all platform implementations. The Application model includes all the currently supported UI objects and their attributes and well as all of the various events that are supported in the default Player. Further, less robust platforms can be augmented by extending higher end capabilities inside that platform's abstraction interface implementation.

In one embodiment, authoring platform 110 provides one or more pages, which may be provided in one Application, or a plurality of Applications, which are stored in memory 123 and subsequently provided to memory 133. In certain embodiments, the Application includes instructions R to request content or web services C from content server 140. Thus, for example and without limitation, the request is for information over the network via a web service, and the request R is responded to with the appropriate information for display on device 130. Thus, for example, a user may request a news report. The Application may include the layout of the display, including a space for the news, which is downloaded from content server 140 for inclusion on the

8

display. Other information that may be provided by content server 140 may include, but is not limited to, pages, Applications, multimedia, and audio.

FIG. 2A is a schematic of a system 200 of an embodiment of system 100 illustrating the communications between different system components. System includes a response director 210, a web component registry 220, and a web service 230. System 200 further includes authoring platform 110, server 120, device 130 and content server 140 are which are generally similar to those of the embodiments of FIGS. 1A and 1B, except as explicitly noted.

Response director 210 is a computer or computer system that may be generally similar to server 120 including the ability to communicate with authoring platform 110 and one or more devices 130. In particular, authoring platform 110 generates one or more Players (each usable by certain devices 130) which are provided to response director 210. Devices 130 may be operated to provide response director 210 with a request for a Player and to receive and install the Player. In one embodiment, device 130 provides response director 210 with device-specific information including but not limited to make, model, and/or software version of the device. Response director 210 then determines the appropriate Player for the device, and provides the device with the Player over the network.

Web service 230 is a plurality of services obtainable over the Internet. Each web service is identified and/or defined as an entry in web component registry 230, which is a database, XML file, or PDL that exists on a computer that may be a server previously described or another server 120. Web component registry 230 is provided through server 120 to authoring platform 110 so that a user of the authoring platform may bind web services 230 to elements to be displayed on device 130, as described subsequently.

In one embodiment, authoring platform 110 is used in conjunction with a display that provides a WYSIWYG environment in which a user of the authoring platform can produce an Application and Player that produces the same display and the desired programming on device 130. Thus, for example, authoring tool 112 provides a display on screen 115 that corresponds to the finished page that will be displayed on screen 137 when an Application is intercepted, via a Player, on processor 135 of device 130.

Authoring platform 110 further permits a user of the authoring platform to associate objects, such as objects for presenting on screen 137, with components of one or more web services 230 that are registered in web component registry 220. In one embodiment, information is provided in an XML file to web component registry 220 for each registered components of each web service 230. Web component registry 220 may contain consumer inputs related to each web service 230, environmental data such as PIM, time or location values, persistent variable data, outputs related to the web service, and/or optional hinting for improving the user's productivity.

A user of authoring platform 110 of system 200 may define associations with web services as WebComponent Bindings. In one embodiment, authoring platform 110 allows a user to associate certain objects for display that provide input or output to components of web service 230. The associated bindings are saved as a PDL in server 120.

In one embodiment, an XML web component registry 220 for each registered web service 230 is loaded into authoring platform 110. The user of system 200 can then assign components of any web service 230 to an Application without any need to write code. In one embodiment, a component of web service 230 is selected from authoring

US 9,928,044 B2

9

platform 110 which presents the user with WYSIWYG dialog boxes that enable the binding of all the inputs and outputs of component of web service 230 to a GUI component of the Application as will be displayed on screen 137. In addition, multiple components of one or more web service 230 can be assigned to any Object or Event in order to facilitate mashups. These Object and/or Event bindings, for each instance of a component of any web service 230, are stored in the PDL. The content server 140 handles all communication between device 130 and the web service 230 and can be automatically deployed as a web application archive to any content server.

Device 130, upon detecting an event in which a component of a web service 230 has been defined, assembles and sends all related inputs to content server 240, which proxies the request to web service 230 and returns the requested information to device 130. The Player on device 130 then takes the outputs of web service 230 and binds the data to the UI components in the Application, as displayed on screen 137.

In one embodiment, the mechanism for binding the outputs of the web service to the UI components is through symbolic references that matches each output to the symbolic name of the UI component. The outputs, in one embodiment, may include meta-data which could become part of the inputs for subsequent interactions with the web service.

For example, if a user of authoring platform 110 wants to present an ATOM feed on device 130, they would search through a list of UI Components available in the authoring platform, select the feed they want to use, and bind the output of the feed summary to a textbox. The bindings would be saved into the PDL on server 120 and processed by device 130 at runtime. If the ATOM feed does not exist a new one can be added to the web component registry that contains all the configuration data required, such as the actual feed URL, the web component manager URL, and what output fields are available for binding.

In another embodiment, components of web services 230 are available either to the user of authoring platform 110 or otherwise accessible through the SDK and Java APIs of routines 114. System 200 permits an expanding set of components of web services 230 including, but not limited to: server pages from content server 120; third-party web services including, but not limited to: searching (such through Google or Yahoo), maps (such as through MapQuest and Yahoo), storefronts (such as through ThumbPlay), SMS share (such as through clickatel), stock quotes, social networking (such as through FaceBook), stock quotes, weather (such as through Accuweather) and/or movie trailers. Other components include web services for communication and sharing through chats and forums and rich messaging alerts, where message alerts are set-up that in turn could have components of Web Services 230 defined within them, including the capture of consumer generated and Web Service supplied rich media and textual content.

System 200 also permits dynamic binding of real-time content, where the inputs and outputs of XML web services are bound to GUI components provided on screen 137. Thus, for example, a user of authoring platform 110 may bind attributes of UI Objects to a particular data base field on a Server. When running the Application, the current value in the referenced data base will be immediately applied. During the Application session, any other real time changes to these values in the referenced data base will again be immediately displayed.

10

As an example of dynamic binding of real-time content, an RSS feeds and other forms of dynamic content may be inserted into mobile Applications, such as device 130, using system 200. Authoring platform 110 may include a "RSS display" list which permits a user to select RSS channels and feeds from an extensible list of available dynamic content. Meta data, such as titles, abstracts and Images can be revealed immediately by the user as they traverse this RSS display list, bringing the PC experience completely and conveniently to mobile devices 130. In addition, Authoring platform 110 may include a dialog box that dynamically links objects to data and feeds determined by RSS and chat databases. Any relevant attribute for a page view and/or object can be dynamically bound to a value in a server-side database. This includes elements within complex objects such as: any icon or text element within a graphical list; any icon within a launch strip; any feature within any geographical view of a GIS service object; and/or any virtual room within a virtual tour.

As an example of third-party web services 230 that may be provided using system 200, a user of authoring platform 110 can place, for example, Yahoo maps into device 130 by binding the required component of the Yahoo Maps Web Service, such as Yahoo Map's Inputs and/or Outputs to appropriate Objects of authoring platform 110. System 200 also provides binding to web services for text, image and video searching by binding to components of those web services.

In one embodiment, an Application for displaying on device 130 includes one or more Applications Pages, each referred to herein as an "XSP," that provides functionality that extends beyond traditional web browsers. The XSP is defined as a PDL, in a similar manner as any Application, although it defines a single page view, and is downloaded to the Player dynamically as required by the PDL definition of the Application. Thus, for example, while JSPs and ASPs, are restricted to the functionality supported by the web browser, the functionality of XSPs can be extended through authoring platform 110 having access to platform dependent routines 114, such as Java APIs. Combined with dynamic binding functionality, an XSP, a page can be saved as a page object in an author's "pages" library, and then can be dynamically populated with real-time content simultaneously as the page is downloaded to a given handset Player based on a newly expanded API. XSP Server Pages can also be produced programmatically, but in most cases authoring platform 110 will be a much more efficient way to generate and maintain libraries of dynamically changing XSPs.

With XSPs, Applications Pages that have dynamic content associated with them can be sent directly to device 130, much like how a web browser downloads an HTML page through an external reference. Without XSPs, content authors would have to define each page in the Application. With XSPs, no pages need to be defined. Thus, for example, in a World Cup Application, one page could represent real-time scores that change continuously on demand. With polling (for example, a prompt to the users asking who they predict will win a game), a back-end database would tabulate the information and then send the results dynamically to the handsets. With a bar chart, the Application would use dynamic PDL with scaling on the fly. For example, the server would recalibrate the bar chart for every ten numbers.

Other combinations of components of web services 230 include, but are not limited to, simultaneous video chat sessions, inside an integrated page view, with a video or television station; multiple simultaneous chat sessions, each

US 9,928,044 B2

11

with a designated individual and/or group, with each of the chat threads visible inside an integrated page view.

Another extension of an XSP is a widget object. Widgets can be developed from numerous sources including, but not limited to, authoring platform 110, a Consumer Publishing Tool, and an XML to Widget Conversion Tool where the SDK Widget Libraries are automatically populated and managed, or Widget Selection Lists that are available and can be populated with author defined Icons.

Applications, Players, and Processing in a Device

FIG. 2B is a schematic of one embodiment of a device 130 illustrating an embodiment of the programming generated by authoring platform 110. Memory 133 may include several different logical portions, such as a heap 133a, a record store 133b and a filesystem (not shown).

As shown in FIG. 2B, heap 133a and record store 133b include programming and/or content. In general, heap 133a is readily accessible by processor 135 and includes, but is not limited to portions that include the following programming: a portion 133a1 for virtual machine compliant objects representing a single Page View for screen 137; a portion 133a2 for a Player; a portion 133a3 for a virtual machine; and a portion 133a4 for an operating system.

Record store 133b (or alternatively the filesystem) includes, but is not limited to, portions 133b1 for Applications and non-streaming content, which may include portions 133a2 for images, portions 133a4 for audio, and/or portions 133a5 for video. and portions 133b2 for non-Application PDLs, such as a Master Page PDL for presenting repeating objects, and Alerts, which are overlayed on the current page view. Other content, such as streaming content may be provided from network interface 131 directly to the Media Codec of device 130 with instructions from Player on how to present the audio or video.

In one embodiment, the Player includes a Threading Model and a Virtual Memory Manager. The Threading Model first manages a queue of actions that can be populated based on Input/Output events, Server-side events, time-based events, or events initiated by user interactions. The Threading Model further manages the simultaneous execution of actions occurring at the same time. The Virtual Memory Manager includes a Logical Virtual Page controller that provides instructions from the record store to the heap, one page at a time. Specifically, the Virtual Memory Manager controls the transfer of one of the Application Pages and its virtual machine compliant objects into portion 133a1 as instructions readable by the Player or Virtual Machine. When the Player determines that a new set of instructions is required, the information (such as one Application Page is retrieve from the Record store, converted into virtual machine compliant objects (by processor 135 and according to operation by the Player, Virtual Machine, etc). and stored in heap 133a. Alternatively, the Player may augment virtual machine compliant objects with its own libraries for managing user interactions, events, memory, etc.

The connection of portions 133a1, 133a2, 133a3, 133a4, record store 133b and processor 135 are illustrative of the logical connection between the different types of programming stored in Heap 133a and record store 133b, that is, how data is processed by processor 135.

The Player determines which of the plurality of Application Pages in portion 133b1 is required next. This may be determined by input actions from the Input Device 139, or from instructions from the current Application Page. The Player instructs processor 135 to extract the PDF from that Applications Page and store it in portion 133a1. The Player then interprets the Application Page extracted from PDL

12

which in turn defines all of the virtual machine compliant Objects, some of which could have attributes that refer to images, audio, and/or video stored in portions 133a3, 133a4, 133a5, respectively.

The Virtual Machine in portion 133a3 processes the Player output, the Operating System in portion 133a3 processes the Virtual Machine output which results in machine code that is processed by the Operating System in portion 133a4.

In another embodiment, the Player is a native program that interacts directly with the operating system.

Embodiments of a Publishing Environment

In one embodiment, authoring platform 110 includes a full-featured authoring tool 112 that provides a what-you-see-is-what-you-get (WYSIWYG) full featured editor. Thus, for example, authoring tool 112 permits a user to design an Application by placing objects on canvas 305 and optionally assigning actions to the objects and save the Application. System 100 then provides the Application and Player to a device 130. The Application as it runs on device 130 has the same look and operation as designed on authoring platform 110. In certain embodiments, authoring platform 110 is, for example and without limitation, a PC-compatible or a Macintosh computer.

Authoring platform 110 produces an Application having one or more Applications Pages, which are similar to web pages. That is, each Applications Page, when executed on device 130 may, according to its contents, modify what is displayed on screen 137 or cause programming on the device to change in a manner similar to how web pages are displayed and navigated through on a website.

In one embodiment, authoring tool 112 allows a user to place one or more objects on canvas 305 and associate the objects with an Applications Pages. Authoring platform 110 maintains a database of object data in memory 111, including but not limited to type of object, location on which page, and object attributes. The user may add settings, events, animations or binding to the object, from authoring tool 112, which are also maintained in memory 111. Authoring tool 112 also allows a user to define more than one Applications Page.

In another embodiment, authoring tool 112, provides Java programming functions of the Java API for specific devices 130 as pull-down menus, dialog boxes, or buttons. This permits a user of authoring platform 110 to position objects that, after being provided as an Application to device 130, activate such Java functions on the device.

In certain embodiments, authoring platform 110, as part of system 100, permits designers to include features of advanced web and web services Applications for access by users of device 130. Some of the features of advanced web and web services include, but are not limited to: slide shows, images, video, audio, animated transitions, multiple chats, and mouse interaction; full 2-D vector graphics; GIS (advanced LBS), including multiple raster and vector layers, feature sensitive interactions, location awareness, streaming and embedded audio/video, virtual tours, image processing and enhancement, and widgets. In other embodiments the features are provided for selection in authoring platform 110 through interactive object libraries.

In certain embodiments, authoring platform 110, as part of system 100, allows the inclusion of child objects which may eventually be activated on device 130 by the user of the device or by time. The uses of the child objects on device 130 include, but are not limited to: mouse over (object selection), hover and fire events and launching of object-specific, rich-media experiences.

US 9,928,044 B2

13

In certain other embodiments, authoring platform **110**, as part of system **100**, provides advanced interactive event models on device **130**, including but not limited to: user-, time- and/or location-initiated events, which allow content developers to base interactivity on specific user interactions and/or instances in time and space; timelines, which are critical for timing of multiple events and for animations when entering, on, or exiting pages of the Application; waypoints, which act similar to key frames, to allow smooth movement of objects within pages of the Application. Waypoints define positions on a page object's animation trajectory. When an object reaches a specific waypoint other object timelines can be initiated, creating location-sensitive multiple object interaction, and/or audio can be defined to play until the object reaches the next waypoint.

Authoring platform **110** may also define a Master Page, which acts as a template for an Applications Page, and may also define Alert Pages, which provide user alerts to a user of device **130**.

In certain embodiments, authoring platform **110**, as part of system **100**, provides full style inheritance on device **130**. Thus, for example and without limitation, both master page inheritance (for structural layout inheritance and repeating objects) and object styles (for both look and feel attribute inheritance) are supported. After a style has been defined for an object, the object will inherit the style. Style attributes include both the look and the feel of an object, including mouse interaction, animations, and timelines. Each page may include objects that may be a parent object or a child object. A child object is one that was created by first selecting a parent object, and then creating a child object. Child objects are always part of the same drawing layer as its parent object, but are drawn first, and are not directly selectable when running the Application. A parent object is any object that is not a child object, and can be selected when running the Application.

As an example, the user of authoring tool **112** may create various text objects on canvas **305** using a style that sets the font to red, the fonts of these objects will be red. Suppose user of authoring tool **112** changes the font color of a specific button to green. If later, the user of authoring tool **112** changes the style to blue; all other text objects that were created with that style will become blue except for the button that had been specifically set to green.

In certain other embodiments, authoring platform **110** provides page view, style, object, widget and Application template libraries. Authoring platform **110** may provide templates in private libraries (available to certain users of the authoring platform) and public libraries (available to all users of the authoring platform). Templates may be used to within authoring platform **110** to define the look and feel of the entire Application, specific pages, or specific slide shows and virtual tours a seen on device **130**.

FIGS. 3A and 3B illustrate one embodiment of a publisher interface **300** as it appears, for example and without limitation, on screen **115** while executing authoring tool **112**. In one embodiment, publisher interface **300** includes a Menu bar **301**, a Tool bar **303**, a Canvas **305**, a Layer Inspector **307** having subcomponents of a page/object panel **307a**, an object style panel **307b**, and a page alert panel **307c**, and a Resource Inspector **309**.

In general, publisher interface **300** permits a user of authoring platform **110** to place objects on canvas **305** and then associate properties and/or actions to the object, which are stored in the Application. As described subsequently, publisher interface **300** permits a user to program a graphical interface for the screen **137** of device **130** on screen **115** of

14

authoring platform **110**, save an Application having the programming instructions, and save a Player for the device. The intended programming is carried out on device **130** when the device, having the appropriate device platform Player, receives and executes the device-independent Application.

Thus, for example, authoring tool **112** maintains, in memory **111**, a list of every type of object and any properties, actions, events, or bindings that may be assigned to that object. As objects are selected for an Application, authoring tool **112** further maintains, in memory **111**, a listing of the objects. As the user selects objects, publisher interface **300** provides the user with a choice of further defining properties, actions, events, or bindings that may be assigned to each particular object, and continues to store the information in memory **111**.

In one embodiment, publisher interface **300** is a graphical interface that permits the placement and association of objects in a manner typical of, for example, vector graphics editing programs (such as Adobe Illustrator). Objects located on canvas **305** placed and manipulated by the various commands within publisher interface **300** or inputs such as an input device **117** which may be a keyboard or mouse. As described herein, the contents of canvas **305** may be saved as an Application that, through system **100**, provide the same or a similar placement of objects on screen **137** and have actions defined within publisher interface **300**. Objects placed on canvas **305** are intended for interaction with user of device **130** and are referred to herein, without limitation, as objects or UI (user interface) objects. In addition, the user of interface **300** may assign or associate actions or web bindings to UI objects placed on canvas **305** with result in the programming device **130** that cause it to respond accordingly.

Objects include, but are not limited to input UI objects, response UI objects. Input UI objects include but are not limited to: text fields (including but not limited to alpha, numeric, phone number, or SMS number); text areas; choice objects (including but not limited to returning the selected visible string or returning a numeric hidden attribute); single item selection lists (including but not limited to returning the selected visible string or returning a numeric hidden attribute); multi item selection lists (including but not limited to returning all selected items (visible text string or hidden attribute) or cluster item selection lists (returning the hidden attributes for all items).

Other input UI objects include but are not limited to: check boxes; slide show (including but not limited to returning a numeric hidden attribute, returning a string hidden attribute, or returning the hidden attributes for all slides); and submit function (which can be assigned to any object including submit buttons, vectors, etc.).

Response UI Objects may include, but are not limited to: single line text objects, which include: a text Field (including but not limited to a URL, audio URL, or purchase URL), a text button, a submit button, or a clear button. Another response UI objects include: a multiple line text object, which may include a text area or a paragraph; a check box; an image; a video; a slide show (with either video or image slides, or both); choice objects; list objects; or control lists, which control all the subordinate output UI objects for that web component. Control list objects include, but are not limited to: list type or a choice type, each of which may include a search response list or RSS display list.

As a further example of objects that may be used with authoring tool **112**, Table I lists Data Types, Preferred Input, Input Candidates, Preferred Output and Output Candidates for one embodiment of an authoring tool.

US 9,928,044 B2

15

16

TABLE I

One embodiment of supported objects				
Data Types	Preferred Input	Input Candidates	Preferred Output	Output Candidates
boolean	Check Box	Check Box	Check Box	Check Box
Int	Text Field (integer)	Text Field (integer) Text Field (Phone #) Text Field (SMS #) Choice List (single select)	Text Field (integer)	Text Field (integer) Text Field (Phone #) Text Field (SMS #) Choice List (single select) Text Button
String	Text Field (Alpha)	Any	Text Field (Alpha)	Any
multilineString	Text Area	Text Area	Text Area	Text Area Paragraph
ImageURL	N/A	N/A	Image	Image Slide Show
VideoURL	N/A	N/A	Video	Video Slide Show
List	Single Item List	Single Item List Multi-Select List Complex List Choice Slide Show	Single Item List	Any List Type Any Choice Type (see Complex List Specification)
ComplexList	Complex List	Single Item List Multi-Select List Complex List	Single Item List	Any List Type (see Complex List Specification)
Slideshow	Slide Show	Slide Show	Slide Show	Slide Show
SearchResponseList	N/A	N/A	Search Response List	Search Response List Control List Complex List Choice RSS Display List Control List Complex List Choice Complex List Multi-Selection List
RSSList	N/A	N/A	RSS Display List	Choice Complex List Choice Complex List Multi-Selection List
SingleSelectionList	Choice	Choice Complex List	Choice	Choice Complex List
MultiSelectionList	Multi-Selection List	Multi-Selection List	Multi-Selection List	Multi-Selection List
ServiceActivation	Submit Button	Any	N/A	N/A
ChannelImageURL	N/A	N/A	Image	Image Video Slide Show Text Area Paragraph Text Field Text Button List Choice Text Field Text Button Paragraph Text Area List Choice Text Field (URL request) Text Field (Audio URL request) Text Field (Purchase URL request) Image Slide Show Slide Show Image N/A N/A
ChannelDescription	N/A	N/A	Text Area	Text Area Paragraph Text Field Text Button List Choice Text Field Text Button Paragraph Text Area List Choice Text Field (URL request) Text Field (Audio URL request) Text Field (Purchase URL request) Image Slide Show Slide Show Image N/A N/A
ChannelTitle	N/A	N/A	Text Field	Text Field (URL request) Text Field (Audio URL request) Text Field (Purchase URL request) Image Slide Show Slide Show Image N/A N/A
URL			Text Field	Text Field (URL request) Text Field (Audio URL request) Text Field (Purchase URL request) Image Slide Show Slide Show Image N/A N/A
Audio URL			Text Field	Text Field (URL request) Text Field (Audio URL request) Text Field (Purchase URL request) Image Slide Show Slide Show Image N/A N/A
Purchase URL			Text Field	Text Field (URL request) Text Field (Audio URL request) Text Field (Purchase URL request) Image Slide Show Slide Show Image N/A N/A
Image Data			Image	Image Slide Show Slide Show Image N/A N/A
Image List Data			Slide Show	Image N/A N/A
Persistent Variable	N/A	N/A	N/A	N/A N/A
Pipeline Multiple Select	Multi-select List	Multi-select List Complex List Slide Show	N/A	N/A
Phone Number	Text Field (numeric type)	Text Field Text Button	Text Field (numeric type)	Text Field Text Button
Hidden Attribute	Complex List	Complex List Slide Show	Complex List	Complex List Slide Show
Collection List	N/A	N/A	Slide Show	Complex List Slide Show

US 9,928,044 B2

17

In general, publisher interface **300** permits a user to define an Application as one or more Applications Pages, select UI objects from Menu bar **301** or Tool bar **303** and arrange them on an Applications Page by placing the objects canvas **305**. An Application Page is a page that is available to be visited through any navigation event. Application Pages inherit all the attributes of the Master Page, unless that attribute is specifically changed during an editing session.

Authoring platform **110** also stores information for each UI object on each Application Page of an Application. Layer Inspector **307** provides lists of Applications Pages, UI objects on each Applications Page, and Styles, including templates. Objects may be selected from canvas **305** or Layer Inspector **307** causing Resource Inspector **309** to provide lists of various UI objects attributes which may be selected from within the Resource Inspector. Publisher interface **300** also permits a user to save their work as an Application for layer transfer and operation of device **130**. Publisher interface **300** thus provides an integrated platform for designing the look and operation of device **130**.

The information stored for each UI object depends, in part, on actions which occur as the result of a user of device **130** selecting the UI object from the device. UI objects include, but are not limited to: navigational objects, such as widget or channel launch strips or selection lists; message objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields or a pop-up alert; text fields or areas; check boxes; pull down menus; selection lists and buttons; pictures; slide shows; video or LBS maps; shapes or text defined by a variety of tools; a search response; or an RSS display.

In certain embodiments, publisher interface **300** permits a user to assign action to UI objects, including but not limited to, programming of the device **130** or a request for information over network N. In one embodiment, for example and without limitation, publisher interface **300** has a selection to bind a UI object to a web service—that is, associate the UI object or a manipulation or selection of UI object with web services. Publisher interface **300** may also include many drawing and text input functions for generating displays that may be, in some ways, similar to drawing and/or word processing programs, as well as toolbars and for zooming and scrolling of a workspace.

Each UI object has some form, color, and display location associate with it. Further, for example and without limitation, UI objects may have navigational actions (such as return to home page), communications actions (such as to call the number in a phone number field), or web services (such as to provide and/or retrieve certain information from a web service). Each of these actions requires authoring platform **110** to store the appropriate information for each action. In addition, UI objects may have associated parent or child objects, default settings, attributes (such as being a password or a phone number), whether a field is editable, animation of the object, all of which may be stored by authoring platform **110**, as appropriate.

Menu bar **301** provides access features of publisher interface **300** through a series of pull-down menus that may include, but are not limited to, the following pull-down menus: a File menu **301a**, an Edit menu **301b**, a View menu **301c**, a Project menu **301d**, an Objects menu **301e**, an Events menu **301f**, a Pages menu **301g**, a Styles menu **301h**, and a Help menu **301i**.

File menu **301a** provides access to files on authoring platform **110** and may include, for example and without limitation, selections to open a new Application or master page, open a saved Application, Application template, or

18

style template, import a page, alert, or widget, open library objects including but not limited to an image, video, slide show, vector or list, and copying an Application to a user or to Server **120**.

Edit menu **301b** may include, but is not limited to, selections for select, cut, copy, paste, and edit functions.

View menu **301c** may include, but is not limited to, selections for zooming in and out, previewing, canvas **305** grid display, and various palette display selections.

Project menu **301d** may include, but is not limited to, selections related to the Application and Player, such as selections that require a log in, generate a universal Player, generate server pages, activate server APIs and extend Player APIs. A Universal Player will include all the code libraries for the Player, including those that are not referenced by the current Application. Server APIs and Player APIs logically extend the Player with Server-side or device-side Application specific logic.

Objects menu **301e** includes selections for placing various objects on canvas **305** including, but not limited to: navigation UI objects, including but not limited to widget or channel launch strips or selection lists; message-related UI objects, including but not limited to multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert; shapes, which provides for drawing tools; forms-related objects, including but not limited to text fields; scrolling text box, check box, drop-down menu, list menu, submit button or clear button; media-related UI objects such as pictures, slide shows, video or LBS maps; text-related UI objects such as buttons or paragraphs; and variables, including but not limited to time, date and audio mute control.

Events menu **301f** includes selections for defining child objects, mouse events, animations or timelines.

Pages menu **301g** includes selection for handling multi-page Applications, and may include selections to set a master page, delete, copy, add or go to Applications Pages.

Styles menu **301h** includes selections to handle styles, which are the underlying set of default appearance attributes or behaviors that define any object that is attached to a style. Styles are a convenient way for quickly creating complex objects, and for changing a whole collection of objects by just modifying their common style. Selections of Styles menu **301h** include, but not limited to, define, import, or modify a style, or apply a template. Help menu **301i** includes access a variety of help topics.

Tool bar **303** provides more direct access to some of the features of publisher interface **300** through a series of pull-down menus. Selections under tool bar **303** may include selections to:

- control the look of publisher interface **300**, such as a Panel selection to control the for hiding or viewing various panels on publisher interface **300**;

- control the layout being designed, such as an Insert Page selection to permit a user to insert and name pages;

- control the functionality of publisher interface **300**, such as a Palettes selection to choose from a variety of specialized palettes, such as a View Palette for zooming and controlling the display of canvas **305**, a Command Palette of common commands, and Color and Shape Palettes;

- place objects on canvas **305**, which may include selections such as: a Navigation selection to place navigational objects, such as widget or channel launch strips or selection lists), a Messages selection to place objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert, a Forms selection to place objects such as text fields or



US 9,928,044 B2

19

areas, check boxes, pull down menus, selection lists, and buttons, a Media selection to place pictures, slide shows, video or LBS maps, and a Shapes selection having a variety of drawing tools, a Text selection for placing text, a search response, or an RSS display, and Palettes.

In one embodiment, Tool bar **303** includes a series of pull-down menus that may include, but are not limited to, items from Menu bar **301** organized in the following pull-down menus: a Panel menu **303a**, an Insert Page menu **303b**, a Navigation menu **303c**, a Messages menu **303d**, a Forms menu **303e**, a Media menu **303f**, a Shapes menu **303g**, a Text menu **303h**, and a Palettes menu **301i**.

Panel menu **303a** permits a user of authoring platform **110** to change the appearance of interface **300** by, controlling which tools are on the interface or the size of canvas **305**. Insert Page menu **303b** permits a user of authoring platform **110** to open a new Application Page. Navigation menu **303c** displays a drop down menu of navigational-related objects such as a widget or channel launch strip or selection list. Messages menu **303d** displays a drop down menu of messaging-related objects such as multiple chat, video chat, phone or SMS lists or fields, and pop-up alerts. Forms menu **303e** displays a drop down menu of forms-related objects including, but not limited to, a text field, a text area, a check box, a drop down menu, a selection list, a submit button, and a clear button. Media menu **303f** displays a drop down menu of media-related objects including, but not limited to, a picture, slide show, video or LBS map. Shapes menu **303g** displays a drop down menu of draw tools, basic shapes, different types of lines and arrows and access to a shape library. Text menu **303h** displays a drop down menu of text-related objects, including but not limited to a text button, paragraph, search response, RSS display and variables such as time and date.

Palettes menu **301i** includes a selection of different palettes that can be moved about publisher interface **300**, where each palette has specialized commands for making adjustments or associations to objects easier. Palettes include, but are not limited to: a page view palette, to permit easy movement between Applications Pages; a view palette, to execute an Application or zoom or otherwise control the viewing of an Application; a commands palette having editing commands; a color palette for selection of object colors; and a shapes palette to facilitate drawing objects.

Layer inspector **307** permits a user of publisher interface **300** to navigate, select and manipulate UI objects on Applications Pages. Thus, for example, a Page/objects panel **307a** of layer inspector **307** has a listing that may be selected to choose an Applications Pages within an Application, and UI objects and styles within an Applications Page. An Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits selection of UI objects for operations to be performed on the objects.

Thus, for example, when objects from Menu bar **301** or Tool bar **303** are placed on canvas **305**, the name of the object appears in Page/objects panel **307a**. Page/objects panel **307a** includes a page display **307a1** and an objects display **307a2**. Page display **307a1** includes a pull down menu listing all Applications Pages of the Application, and objects display **307a2** includes a list of all objects in the Applications Page (that is, objects on canvas **305**).

In general, page/objects panel **307a** displays various associations with a UI object and permits various manipulations including, but not limited to, operations for parent and child objects that are assigned to a page, and operations for object styles, and permits navigating between page types

20

and object styles, such as switching between the master page and Application pages and deselecting object styles and alerts, opening an Edit Styles Dialog Box and deselecting any master, Application or alert page, or selecting an alert page and deselecting any Master Page or Application Page. A parent or child object can also be selected directly from the Canvas. In either case, the Resource Inspector can then be used for modifying any attribute of the selected object.

Examples of operations provided by page/objects panel **307a** on pages include, but are not limited to: importing from either a user's private page library or a public page library; deleting a page; inserting a new page, inheriting all the attributes of the Master Page, and placing the new page at any location in the Page List; editing the currently selected page, by working with an Edit Page Dialog Box. While editing all the functions of the Resource Inspector **309** are available, as described subsequently, but are not applied to the actual page until completing the editing process.

Examples of operations provided by of page/objects panel **307a** on objects, which may be user interface (UI) objects, include but are not limited to: changing the drawing order layer to: bring to the front, send to the back, bring to the front one layer, or send to the back one layer; hiding (and then reshown) selected objects to show UI objects obstructed by other UI Objects, delete a selected UI Page Object, and editing the currently selected page, by working with a Edit Page Dialog Box.

Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits operations to be performed on objects, and is similar to panel **307a**. Examples of operations provided by object style panel **307b** include, but are not limited to: importing from either a user's private object library or a public object library; inserting a new object style, which can be inherited from a currently selected object, or from a previously defined style object; and editing a currently selected object style by working with an Edit Style Dialog Box.

Style attributes can be assigned many attributes, including the look, and behavior of any object that inherits these objects. In addition, List Layout Styles can be created or changed as required. A layout style can define an unbounded set of Complex List Layouts, including but not limited to: the number of lines per item in a list, the number of text and image elements and their location for each line for each item in the last, the color and font for each text element, and the vertical and horizontal offset for each image and text element.

Alerts Panel **307c** provides a way of providing alert pages, which can have many of the attributes of Application Pages, but they are only activated through an Event such as a user interaction, a network event, a timer event, or a system variable setting, and will be superimposed onto whatever is currently being displayed. Alert Pages all have transparent backgrounds, and they function as a template overlay, and can also have dynamic binding to real time content.

Resource inspector **309** is the primary panel for interactively working with UI objects that have been placed on the Canvas **305**. When a UI object is selected on Canvas **305**, a user of authoring platform **110** may associate properties of the selected object by entering or selecting from resource inspector **309**. In one embodiment, resource inspector **309** includes five tab selections: Setting Tab **309a**, Events Tab **309b**, Animation Tab **309c**, Color Tab **309d** which includes a color palette for selecting object colors, and Bindings Tab **309e**.

Settings Tab **309a** provides a dialog box for the basic configuration of the selected object including, but not limited to:

US 9,928,044 B2

21

ited to, name, size, location, navigation and visual settings. Depending upon the type of object, numerous other attributes could be settable. As an example, the Setting Tab for a Text Field may include dialog boxes to define the text field string, define the object style, set the font name, size and effects, set an object name, frame style, frame width, text attributes (text field, password field, numeric field, phone number, SMS number, URL request).

As an example of Setting Tab **309a**, FIG. 3B shows various selections including, but not limited to, setting **309a1** for the web page name, setting **309a2** for the page size, including selections for specific devices **130**, setting **309a3** indicating the width and height of the object, and setting **309a4** to select whether background audio is present and to select an audio file.

FIG. 3C illustrates an embodiment of the Events Tab **309b**, which includes all end user interactions and time based operations. The embodiment of Events Tab **309b** in FIG. 3C includes, for example and without limitation, an Events and Services **309b1**, Advanced Interactive Settings **309b2**, Mouse State **309b3**, Object Selected Audio Setting **309b4**, and Work with Child Objects and Mouse Overs button **309b5**.

Events and Services **309b1** lists events and services that may be applied to the selected objects. These include, but are not limited to, going to external web pages or other Applications pages, either as a new page or by launching a new window, executing an Application or JavaScript method, pausing or exiting, placing a phone call or SMS message, with or without single or multiple Player download, show launch strip, or go back to previous page. Examples of events and services include, but are not limited to those listed in Table II

TABLE II

Events and Services	
Goto External Web Page replacing Current Frame	ChoiceObject: Remove Icon from Launch Strip
Goto External Web Page Launched in a New Window	Goto a specific Internal Web Page with Alert. "Backend Synchronization"
Goto a specific Internal Web Page	Goto Widget Object
Goto the next Internal Web Page	Generate Alert. "With a Fire Event"
Goto External Web Page replacing the Top Frame	Send SMS Message from Linked Text Field
Execute JavaScript Method	Toggle Alert. "Display OnFocus, Hide OffFocus"
Pause/Resume Page Timeout	Execute an Application with Alert. "With a Fire Event"
Execute an Application	Goto Logical First Page
Goto a specific Internal Web Page with setting starting slide	Generate Alert with Backend Synchronization
Exit Application	Send SMS Message with Share (Player Download)
Exit Player	Place PhoneCall from linked Text Field with Share (Player Download)
Place PhoneCall from linked Text Field	Send IM Alert from linked Text Field or Text Area
Text Field/Area: Send String on FIRE	Set and Goto Starting Page
ChoiceObject: Add Icon to Launch Strip	Populate Image
Text Field/Area: Send String on FIRE or Numeric Keys	Preferred Launch Strip

Advanced Interactive Settings **309b2** include Scroll Activation Enabled, Timeline Entry Suppressed, Enable Server Listener, Submit Form, Toggle Children on FIRE, and Hide Non-related Children, Mouse State **309b3** selections are Selected or Fire. When Mouse State Selected is chosen,

22

Object Selected Audio Setting **309b4** of Inactive, Play Once, Loop, and other responses are presented. When Mouse State Fire is chosen, Object Selected Audio Setting **309b4** is replaced with FIRE Audi Setting, with appropriate choices presented.

When Work with Child Objects and Mouse Overs button **309b5** is selected, a Child Object Mode box pops up, allowing a user to create a child object with shortcut to Menu bar **301** actions that may be used define child objects.

FIG. 3D illustrates one embodiment of an Animation Tab **309c**, which includes all animations and timelines. The Color Tab includes all the possible color attributes, which may vary significantly by object type.

Animation Tab **309c** includes settings involved in animation and timelines that may be associated with objects. One embodiment of Animation Tab **309c** is shown, without limitation, in FIG. 3D, and is described, in Rempell ("Rempell").

A Color Tab **309d** includes a color palette for selecting object colors.

Bindings Tab **309e** is where web component operations are defined and dynamic binding settings are assigned. Thus, for example, a UI object is selected from canvas **305**, and a web component may be selected and configured from the bindings tab. When the user's work is saved, binding information is associated with the UI object that will appear on screen **137**.

FIG. 3E illustrates one embodiment of Bindings Tab and includes, without limitation, the following portions: Web Component and Web Services Operations **309e1**, Attributes Exposed list **309e2**, panel **309e3** which includes dynamic binding of server-side data base values to attributes for the selected object, Default Attribute Value **309e4**, Database Name **309e5**, Table Name **309e6**, Field Name **309e7**, Channel Name **309e8**, Channel Feed **309e9**, Operation **309e10**, Select Link **309e11**, and Link Set checkbox **309e12**.

Web Component and Web Services Operations **309e1** includes web components that may be added, edited or removed from a selected object. Since multiple web components can be added to the same object, any combination of mash-ups of 3rd party web services is possible. When the "Add" button of Web Component and Web Services Operations **309e1** is selected, a pop-up menu **319**, as shown in FIG. 3F, appears on publisher interface **300**. Pop-up menu **319** includes, but is not limited to, the options of: Select a Web Component **319a**; Select Results Page **319b**; Activation Options **319c**; Generate UI Objects **319d**; and Share Web Component **319e**.

The Select a Web Component **319a** portion presents a list of web components. As discussed herein, the web components are registered and are obtained from web component registry **220**.

Select Results Page **319b** is used to have the input and output on different pages—that is, when the Results page is different from Input page. The default selected results page is either the current page, or, if there are both inputs and outputs, it will be set provisionally to the next page in the current page order, if one exists.

Activation Options **319c** include, if there are no Input UI Objects, a choice to either "Preload" the web component, similar to how dynamic binding, or have the web component executed when the "Results" page is viewed by the consumer.

Generate UI Objects **319d**, if selected, will automatically generate the UI objects. If not selected, then the author will bind the Web Component Inputs and Results to previously created UI Objects.

US 9,928,044 B2

23

Share Web Component **319e** is available and will become selected under the following conditions: 1) Web Component is Selected which already has been used by the current Application; or 2) the current Input page is also a "Result" page for that Web component. This permits the user of device **130**, after viewing the results, to extend the Web Component allowing the user to make additional queries against the same Web Component. Examples of this include, but are not limited to, interactive panning and zooming for a Mapping Application, or additional and or refined searches for a Search Application.

Dynamic Binding permits the binding of real time data, that could either reside in a 3<sup>rd</sup> party server-side data base, or in the database maintained by Feed Collector **1010** for aggregating live RSS feeds, as described subsequently with reference to FIG. **10**.

Referring again to FIG. **3E**, Attributes Exposed list **309e2** are the attributes available for the selected object that can be defined in real time through dynamic binding.

Panel **309e3** exposes all the fields and tables associated with registered server-side data bases. In one embodiment, the user would select an attribute from the "Attributes Exposed List" and then select a data base, table and field to define the real time binding process. The final step is to define the record. If the Feed Collector data base is selected, for example, then the RSS "Channel Name" and the "Channel Feed" drop down menus will be available for symbolically selected the record. For other data bases the RSS "Channel Name" and the "Channel Feed" drop down menus are replaced by a "Record ID" text field.

Default Attribute Value **309e4** indicates the currently defined value for the selected attribute. It will be overridden in real time based on the dynamic linkage setting.

Database Name **309e5** indicates which server side data base is currently selected.

Table Name **309e6** indicates which table of the server side data base is currently selected.

Field Name **309e7**, indicates which field form the selected table of the server side data base is currently selected.

Channel Name **309e8** indicates a list of all the RSS feeds currently supported by the Feed Collector. This may be replaced by "Record ID" if a data base other than the Feed Collector **1010** is selected.

Channel Feed **309e9** indicates the particular RSS feed for the selected RSS Channel. Feed Collector **1010** may maintain multiple feeds for each RSS channel.

Operation **309e10**, as a default operation, replaces the default attribute value with the real time value. In other embodiments this operation could be append, add, subtract, multiply or divide.

Select Link **309e11** a button that, when pressed, creates the dynamic binding. Touching the "Select Link" will cause the current data base selections to begin the blink is some manner, and the "Select Link" will change to "Create Link". The user could still change the data base and attribute choices. Touching the "Create Link" will set the "Link Set" checkbox and the "Create Link" will be replaced by "Delete Link" if the user wishes to subsequently remove the link. When the application is saved, the current active links are used to create the SPDL.

Link Set checkbox **309e12** indicates that a link is currently active.

An example of the design of a display is shown in FIGS. **4A** and **4B** according the system **100**, where FIG. **4A** shows publisher interface **300** having a layout **410** on canvas **305**, and FIG. **4B** shows a device **130** having the resulting layout **420** on screen **137**. Thus, for example, authoring platform

24

**110** is used to design layout **410**. Authoring platform **110** then generates an Application and a Player specific to device **130** of FIG. **4B**. The Application and Player are thus used by device **130** to produce layout **420** on screen **137**.

As illustrated in FIG. **4A**, a user has placed the following on canvas **305** to generate layout **410**: text and background designs **411**, a first text input box **413**, a second text input box **415**, and a button **417**. As an example which is not meant to limit the scope of the present invention, layout **410** is screen prompts a user to enter a user name in box **413** and a password in box **415**, and enter the information by clicking on button **417**.

In one embodiment, all UI objects are initially rendered as Java objects on canvas **305**. When the Application is saved, the UI objects are transformed into the PDL, as described subsequently.

Thus, for example, layout **410** may be produced by the user of authoring platform **110** selecting and placing a first Text Field as box **413** then using the Resource Inspector **309** portion of interface **300** to define its attributes. Device User Experience

Systems **100** and **200** provide the ability for a very large number of different types of user experiences. Some of these are a direct result of the ability of authoring platform **110** to bind UI objects to components of web services. The following description is illustrative of some of the many types of experiences of using a device **130** as part of system **100** or **200**.

Device **130** may have a one or more of a very powerful and broad set of extensible navigation objects, as well as object- and pointer-navigation options to make it easy to provide a small mobile device screen **137** with content and to navigate easily among page views, between Applications, or within objects in a single page view of an Application.

Navigation objects include various types of launch strips, various intelligent and user-friendly text fields and scrolling text boxes, powerful graphical complex lists, as well as Desktop-level business forms. In fact, every type of object can be used for navigation by assigning a navigation event to it. The authoring tool offers a list of navigation object templates, which then can be modified in numerous ways. Launch Strips and Graphical List Templates Launch Strips

Launch strips may be designed by the user of authoring platform **110** with almost no restrictions. They can be stationary or appear on command from any edge of the device, their size, style, audio feedback, and animations can be freely defined to create highly compelling experiences.

FIG. **5** shows a display **500** of launch strips which may be on displayed canvas **305** or on screen **137** of device **130** having the proper Player and Application. Display **501** includes a portal-type Launch Strip **501** and a channel-type Launch Strip **502**, either one of which may be included for navigating the Application.

Launch Strip **501** includes UI objects **501a**, **501b**, **501c**, **501d**, and **501e** that that becomes visible from the left edge of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector **309** with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. When the Applications Page, having been saved by authoring platform **110** and transferred to display **130**, is executed on device **130**, a user of the device may easily navigate the Application.

Launch Strip **502** includes UI objects **502b**, **502c**, **502d**, and **503e** that that becomes visible from the bottom of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector

US 9,928,044 B2

25

309 with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. Launch Strip 502 also includes UI objects 502a and 503g, which include the graphic of arrows, and which provide access to additional navigation objects (not shown) when selected by a user of device 130. Launch strip 502 may also include sound effects for each channel when being selected, as well as popup bubble help.

Additional navigational features are illustrated in FIG. 6A as a display of a Channel Selection List 601a, in FIG. 6B as a display of a Widget Selection List 601b, and in FIG. 6C as a display of a Phone List 601c. Lists 601a, 601b, and 601c may be displayed on canvas 305 or on screen 137 of device 130 having the proper Player and Application. As illustrated, graphical lists 601a, 601b, and 601c may contain items with many possible text and image elements. Each element can be defined at authoring time and/or populated dynamically through one or more Web Service 250 or API. Assignable Navigation Events. All objects, and/or all elements within an object, can be assigned navigation events that can be extended to registered web services or APIs. For example, a Rolodex-type of navigation event can dynamically set the starting slide of the targeted page view (or the starting view of a targeted Application).

In the embodiment of FIGS. 6A, 6B, and 6C, each list 601a, 601b, and 601c has several individual entries that are each linked to specific actions. Thus Channel Selection List 601a shows three objects, each dynamically linked to a web service (ESPN, SF 49ers, and Netflix) each providing a link to purchase or obtain items from the Internet. Widget Selection List 601b includes several objects presenting different widgets for selecting. Phone List 601c includes a list phone number objects of names that, when selected by a user of device 130 cause the number to be dialed Entries in Phone List 601c may be generated automatically from either the user's contact list that is resident on the device, or through a dynamic link to any of user's chosen server-side facilities such as Microsoft Outlook, Google Mail, etc. In one embodiment, Phone List 601c may be generated automatically using a web component assigned to the Application, which would automatically perform those functions.

In another embodiment, authoring platform 110 allows a navigation selection of objects with a Joy Stick and/or Cursor Keys in all 4 directions. When within a complex object the navigation system automatically adopts to the navigation needs for that object. For coordinate sensitive objects such as geographical information services (GIS) and location-based services (LBS) or virtual tours a soft cursor appears. For Lists, scrolling text areas and chats, Launch strips, and slide shows the navigation process permits intuitive selection of elements within the object. Scroll bars and elevators are optionally available for feedback. If the device has a pointing mechanism then scroll bars are active and simulate the desktop experience.

Personalization and Temporal Adoption

System 100 and 200 permit for the personalization of device 130 by a variety of means. Specifically, what is displayed on screen 137 may depend on either adoption or customization. Adoption refers to the selection of choices, navigation options, etc. are based on user usage patterns. Temporal adoption permits the skins, choices, layouts, content, widgets, etc. to be further influenced by location (for example home, work or traveling) and time of day (including season and day of week). Customization refers to user selectable skins, choices, layouts, dynamic content, widgets, etc. that are available either through a customization on the

26

phone or one that is on the desktop but dynamically linked to the user's other internet connected devices.

To support many personalization functions there must be a convenient method for maintaining, both within a user's session, and between sessions, memory about various user choices and events. Both utilizing a persistent storage mechanism on the device, or a database for user profiles on a server, may be employed.

FIG. 7 shows a display 700 of a mash-up which may be on displayed canvas 305 or on screen 137 of device 130 having the proper Player and Application. Display 700 includes several object 701 that have been dynamically bound, including an indication of time 701a, an indication of unread text messages 701b, an RSS news feed 701c (including 2 "ESPN Top Stories" 701c1 and 701c2), components 701d from two Web Services—a weather report ("The Weather Channel"), and a traffic report 701e ("TRAFFIC.COM").

In assembling the information of display 700, device 130 is aware of the time and location of the device—in this example the display is for a workday when a user wakes. Device 130 has been customized so that on a work day morning the user wishes to receive the displayed information. Thus in the morning, any messages received overnight would be flagged, the user's favorite RSS sports feeds would be visible, today's weather forecast would be available, and the current traffic conditions between the user's home and office would be graphically depicted. User personalization settings may be maintained as persistent storage on device 130 when appropriate, or in a user profile which is maintained and updated in real-time in a server-side data base. Push Capable Systems

In another embodiment system 100 or 200 is a push-capable system. As an example, of such systems, short codes may be applied to cereal boxes and beverage containers, and SMS text fields can be applied to promotional websites. In either case, a user of device 130 can text the short code or text field to an SMS server, which then serves the appropriate Application link back to device 130.

FIG. 8 is a schematic of an embodiment of a push enabled system 800. System 800 is generally similar to system 100 or 200. Device 130 is shown as part of a schematic of a push capable system 800 in FIG. 8. System 800 includes a website system 801 hosting a website 801, a server 803 and a content server 805. System 801 is connected to servers 803 and/or 805 through the Internet. Server 803 is generally similar to server 120, servers 805 is generally similar to server 140.

In one embodiment, a user sets up a weekly SMS update from website system 801. System 801 provides user information to server 803, which is an SMS server, when an update is ready for delivery. Server 803 provides device 130 with an SMS indication that the subscribed information is available and queries the user to see if they wish to receive the update. Website 801 also provides content server 805 with the content of the update. When a user of device 130 responds to the SMS query, the response is provided to content server 805, which provides device 130 with updates including the subscribed content.

In an alternative embodiment of system 800, server 803 broadcasts alerts to one or more devices 130, such as a logical group of devices. The user is notified in real-time of the pending alert, and can view and interact with the message without interrupting the current Application.

FIG. 9 is a schematic of an alternative embodiment of a push enabled system 900. System 900 is generally similar to system 100, 200, or 800. In system 900 a user requests information using an SMS code, which is delivered to device

US 9,928,044 B2

27

130. System 900 includes a promotional code 901, a third-party server 903, and content server 805. Server 803 is connected to servers 803 and/or 805 through the Internet, and is generally similar to server 120.

A promotional code 901 is provided to a user of device 130, for example and without limitation, on print media, such as on a cereal box. The use of device 130 sends the code server 903. Server 903 then notifies server 805 to provide certain information to device 130. Server 805 then provides device 130 with the requested information.

#### Device Routines

Device routines 114 may include, but are not limited to: an authoring tool SDK for custom code development including full set of Java APIs to make it easy to add extensions and functionality to mobile Applications and tie Applications to back-end databases through the content server 140; an expanding set of web services 250 available through the authoring tool SDK; a web services interface to SOAP/XML enabled web services; and an RSS/Atom and RDF feed collector 1010 and content gateway 1130.

#### Authoring Tool SDK for Custom Code Development Including Full Set of Java APIs

In one embodiment, authoring platform 110 SDK is compatible for working with various integrated development environments (IDE) and popular plug ins such as J2ME Polish. In one embodiment the SDK would be another plug in to these IDEs. A large and powerful set of APIs and interfaces are thus available through the SDK to permit the seamless extension of any Application to back end business logic, web services, etc. These interfaces and APIs may also support listeners and player-side object operations.

There is a large set of listeners that expose both player-side events and dynamically linked server side data base events. Some examples of player side events are: player-side time based event, a page entry event, player-side user interactions and player-side object status. Examples of server-side data base events are when a particular set of linked data base field values change, or some field value exceeds a certain limit, etc.

A superset of all authoring tool functionality is available through APIs for layer-side object operations. These include, but are not limited to: page view level APIs for inserting, replacing, and or modifying any page object; Object Level APIs for modifying any attribute of existing objects, adding definitions to attributes, and adding, hiding or replacing any object.

#### Authoring Tool SDK Available Web Services

The APIs permit, without limit, respond, with or without relying on back-end business logic, that is, logic that what an enterprise has developed for their business, to any player-side event or server-side dynamically linked data-base, incorporating any open 3rd party web service(s) into the response.

#### RSS/ATOM and RDF Feed Conversion Web Service

FIG. 10 is a schematic of one embodiment a system 1000 having a feed collector 1010. System 1000 is generally similar to system 100, 200, 800, or 900. Feed collector 1010 is a server side component of system 100 that collects RSS, ATOM and RDF format feeds from various sources 1001 and aggregates them into a database 1022 for use by the Applications built using authoring platform 110.

Feed collector 1010 is a standard XML DOM data extraction process, and includes Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, and Custom Populator Rule 1016, DOM XML Parsers 1011, 1015, and 1017, Feed Processed Data Writer 1018, Custom Rule

28

Based Field Extraction 1019, Rule-based Field Extraction 1020, Channel Data Controller 1021, and Database 1022.

The feed collector is primarily driven by two sets of parameters: one is the database schema (written as SQL DDL) which defines the tables in the database, as well as parameters for each of the feeds to be examined. The other is the feed collection rules, written in XML, which can be used to customize the information that is extracted from the feeds. Each of the feeds is collected at intervals specified by the feed parameter set in the SQL DDL.

Feed collector 1010 accepts information from ATOM, RDF or RSS feed sources 1001. Using a rules-based populator, any of these feeds can be logically parsed, with any type of data extraction methodology, either by using supplied rules, or by the author defining their own custom extraction rule. The rules are used by the parser to parse from the feed sources, and the custom rule base field extraction replaces the default rules and assembles the parsed information into the database.

In particular, Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, Custom Populator Rule 1016, and DOM XML Parsers 1011, 1015, and 1017 are parse information from the feeds 1001, and Feed Processed Data Writer 1018, Custom Rule Based Field Extraction 1019, Rule-based Field Extraction 1020, and Channel Data Controller 1021, supply the content of the feeds in Database 1022, which is accessible through content server 140.

FIG. 11 is a schematic of an embodiment of a system 1100 having a Mobile Content Gateway 1130. System 1100 is generally similar to system 100, 200, 800, 900, or 1000. System 1100 includes an SDK 1131, feed collector 1010, database listener 1133, transaction server 1134, custom code 1135 generated from the SDK, Java APIs, Web Services 1137, and PDL snippets compacted objects 1139. System 1100 accepts input from Back End Java Code Developer 1120 and SOAP XML from Web Services 1110, and provides dynamic content to server 140 and Players to devices 130.

In one embodiment authoring platform 110 produces a Server-side PDL (SPDL) at authoring time. The SPDL resides in server 120 and provides a logical link between the Application's UI attributes and dynamic content in database 1022. When a user of device 130 requests dynamic information, server 120 uses the SPDL to determine the link required to access the requested content.

In another embodiment Web Services 1137 interface directly with 3rd party Web Services 1110, using SOAP, REST, JAVA, JavaScript, or any other interface for dynamically updating the attributes of the Application's UI objects. XSP Web Pages as a Web Service

In one embodiment, a PDL for a page is embedded within an HTML shell, forming one XSP page. The process of forming XSP includes compressing the description of the page and then embedding the page within an HTML shell.

In another embodiment, a PDL, which contains many individual page definitions, is split into separate library objects on the server, so that each page can be presented as a PDL as part of a Web Service.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java), and The code and data have been reduced by 4 to 10 times.

Compression has two distinct phases. The first takes advantage of how the primitive representations had been assembled, while the second utilizes standard LZ encoding.

US 9,928,044 B2

29

The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

One embodiment for compacting data that may be used is described in Rempell. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

In Rempell, a process for compacting a "database" (that is, generating a compact PDL) is described, wherein data objects, including but not limited to, multi media objects such as colors, fonts, images, sound clips, URLs, threads, and video, including multi level animation, transformation, and time line are compacted. As an extension to Rempell in all cases these objects are reduced and transformed to Boolean, integer and string arrays.

The compression technique involves storing data in the smallest arrays necessary to compactly store web page information. The technique also includes an advanced form of delta compression that reduces integers so that they can be stored in a single byte, as high water marks.

Thus, for example, the high water mark for different types of data comprising specific web site settings are stored in a header record as Boolean and integer variables and URL and color objects. Data that defines web page, paragraph, text button, and image style and text button, image and paragraph high watermark settings can be stored in one-dimensional arrays as Boolean, integer and string variables and URL, font, image or thread objects at. The URL, color, font, image and thread objects can also be created as required.

Data that defines text button, image, paragraph, or other parent objects and paragraph line high watermark settings can be stored in two-dimensional arrays (by web page and by object number) as Boolean, integer, string, floating point variables and URLs. Again, the URL, color, font, image, audio clip, video clip, text area and thread objects can also be created as required. Data that defines a paragraph line and paragraph line segment high watermarks can be stored in three-dimensional arrays (by web page, by paragraph number, and by line number) as Boolean, integer or string variables. Again, the URL, color or font objects can be created as required. Data that defines a paragraph line segment can be stored into four-dimensional arrays (by web page, by paragraph number, by line number and by line number segment) as Boolean, integer or string variables or URL, color and font objects.

As a data field is added, changed or deleted, a determination is made at on whether a value for a given high watermark needs to be changed. If so, it is updated. As a specific method in the build engine is called, a determination is made on whether a feature flag needs to be set. For example, if a particular JAVA method is called, which requires an instance of a certain JAVA Class to be executed by the run time engine, then that JAVA Class is flagged, as well as any supporting methods, variables and/or object definitions.

In one implementation, the header record, the style record, the web page record, and the object records, are carefully defined in a specific order, written in that order, and explicitly cast by object type when read by the run time engine. Exception handling can be implemented to recover from any errors. This helps assure that data integrity is maintained throughout the build and run time processes.

Also described in Rempell is the "run generation process." This is equivalent generating a Player in the present application. This process starts when the build process detects that the user is finished defining the web site (user has saved the web site and invokes the run generation

30

process), and concludes with the actual uploading of all the necessary web site run time files to the user's server.

In one embodiment, the PDL includes a first record, a "Header" record, which contains can include the following information:

1: A file format version number, used for upgrading database in future releases.

2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user.

3: Whether the Application is a web site.

4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.

5: Web page and styles high watermarks.

6: The Websitename.

As new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the PDL, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

The settings for all paragraph, text button and image styles are then written as a style record based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist.

The body of the PDL is then written. All Boolean values are written inside a four-dimensional loop. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects (i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of line segments per paragraph line and contains the values

US 9,928,044 B2

31

used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment).

All integer values are written inside a four-dimensional loop. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

Multimedia objects are written inside a two-dimensional loop. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a web browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other objects on a web page.

String records are written inside a four-dimensional loop. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The third loop contains values for paragraph line related data, and the innermost loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized.

Single and double floating-point, and long integer records are written inside a two-dimensional loop. The outer loop may be empty. The inner loop contains mathematical values required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

In one embodiment, a versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation.

All external image, video and audio files are resolved. The external references can be copied to designated directories, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries are either installed on the local system or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code. Finally, the run time engine for the web site is created. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file.

Next, an HTML Shell File (HSF) is constructed. The first step of this process is to determine whether the dynamic web page and object resizing is desired by testing the Application setting. If the Application was a web page, and thus requiring dynamic web page and object resizing, virtual screen

32

resolution settings are placed in an appropriate HTML compliant string. If the Application is a banner or other customized Application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values.

An analysis is made for the background definition for the first internal web page. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the web browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a web browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the web browser. More specifically, if the Application required dynamic web page and object resizing then JavaScript and HTML compliant strings are generated so that, when interpreted by the web browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated in order to enable JavaScript to SRS applet communication. In one implementation, the code is generated by performing the following functions:

- 1: Determine the current web browser type.
- 2: Load the SRS from either a JAR or CAB File, based on web browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.
- 5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated that perform the following steps at the time the HSF is initialized by the web browser:

- 1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.
- 2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the PDL.
- 3: Generate an HTML complaint string, dependent upon the type of web browser, which causes the current web browser to load either the JAR or the CAB File(s).
- 4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the web browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

The writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "Website-name".html file is created.

In one embodiment, the processes for creating the CAB and JAR Files is as follows. The image objects, if any, which were defined on the first internal web page are analyzed. If they are set to draw immediately upon the loading of the first web page, then they are flagged for compression and inclu-

US 9,928,044 B2

33

sion in the CAB and JAR Files. The feature flags are analyzed to determine which JAVA classes have been compiled. These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "Website-name".class, customized run time engine file, and the "Website-name".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "Website-name".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "Website-name".bat and "Website-namelib".bat files are written. The "Website-name".bat and "Website-name".bat files are then executed under DOS, creating compressed "Website-name".cab and "Website-namelib".cab files and compressed "Website-name" jar and "Website-namelib" jar files. The HTML Shell File and the JAR and CAB files are then, either as an automatic process, or manually, uploaded to the user's web site. This completes the production of an XSP page that may be accessed through a web browser.

#### Displaying Content on a Device Decompression Management

Authoring platform 110 uses compaction to transform the code and data in an intelligent way while preserving all of the original classes, methods and attributes. This requires both an intelligent server engine and client (handset) Player, both of which fully understand what the data means and how it will be used.

The compaction technology described above includes transformation algorithms that deconstruct the logic and data into their most primitive representations, and then reassembles them in a way that can be optimally digested by further compression processing. This reassembled set of primitive representations defines the PDL of authoring platform 110.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java). The data is then compressed by first taking advantage of how the primitive representations had been assembled, and then by utilizing standard LZ encoding. The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

The Player, when preparing a page view for execution, decompresses and then regenerate the original objects, but this time in compliance with the programming APIs of device 130. Specifically, device 130 operates on compacted image pages, one at a time. The cache manager retrieves, decompresses, and reassembles the compacted page images into device objects, which are then interpreted by device 130 for display on screen 137.

#### Response Director

In one embodiment, system 100 includes a Response Director, which determines a user's handset, fetches the correct Application from different databases, and delivers a respective highly compressed Application in a PDL format over the air (OTA).

34

In one embodiment, the Response Director operates on a network connected computer to provide the correct Player to a given device based on the information the device sent to it. As an example, this may occur when a device user enters their phone number into some call-to-action web page. The response director is called and sends an SMS message to the device, which responds, beginning the recognition process.

FIG. 12 illustrates one embodiment of a system 1200 that includes a response director 210, a user agent database 1201, an IP address database 1203, and a file database 1205. System 1200 is generally similar to system 100, 200, 800, 900, 1000, or 1100.

Databases 1201, 1203, and 1205 may reside on server 120, 210, or any computer system in communication with response director 210. System 1200, any mobile device can be serviced, and the most appropriate Application for the device will be delivered to the device, based on the characteristics of the device.

User agent database 1201 includes user agent information regarding individual devices 130 that are used to identify the operating system on the device. IP address database 1203 identifies the carrier/operator of each device 130. File database 1205 includes data files that may operate on each device 130.

The following is an illustrative example of the operation of response director 210. First, a device 1300 generates an SMS message, which automatically sends an http:// stream that includes handset information and its phone number to response director 210. Response director 210 then looks at a field in the http header (which includes the user agent and IP address) that identifies the web browser (i.e., the "User Agent"). The User Agent prompts a database lookup in user agent database 1201 which returns data including, but not limited to, make, model, attributes, MIDP 1.0 MIDP 2.0, WAP and distinguishes the same models from different countries. A lookup of the IP address in IP address 1203 identifies the carrier/operator.

File database 1205 contains data types, which may include as jad1, jad2, html, wml/wap2, or other data types, appropriate for each device 130. A list of available Applications are returned to a decision tree, which then returns, to device 130, the Application that is appropriate for the respective device. For each file type, there is an attributes list (e.g., streaming video, embedded video, streaming audio, etc.) to provide enough information to determine what to send to the handset.

Response director 210 generates or updates an html or jad file populating this text file with the necessary device and network dependent parameters, including the Application dependent parameters, and then generate, for example, a CAB or JAD file which contains the necessary Player for that device. For example, the jad file could contain the operator or device type or extended device-specific functions that the player would then become aware of

If there is an Application that has a data type that device 130 cannot support, for example, video, response director 210 sends an alternative Application to the handset, for example one that has a slide show instead. If the device cannot support a slide show, an Application might have text and images and display a message that indicates it does not support video.

Another powerful feature of response director 210 is its exposed API from the decision tree that permits the overriding of the default output of the decision tree by solution providers. These solution providers are often licensees who want to further refine the fulfillment of Applications and Players to specific devices beyond what the default algo-



US 9,928,044 B2

35

gorithms provide. Solution providers may be given a choice of Applications and then can decide to use the defaults or force other Applications.

Authoring platform 110 automatically scales Applications at publishing time to various form factors to reduce the amount of fragmentation among devices, and the Response Director serves the appropriately scaled version to the device. For example, a QVGA Application will automatically scale to the QCIF form factor. This is important because one of the most visible forms of fragmentation resides in the various form factors of wireless, and particularly mobile, devices, which range from 128×128, 176×208, 240×260, 220×220, and many other customized sizes in between.

FIG. 13 is a schematic of an embodiment of a system 1300. System 1300 is generally similar to system 1200. System 1300 is an overview of the entire Player fulfillment process, starting with the generation of players during the player build process.

System 1300 includes response director 210, a device characteristics operator and local database 1301, a player profile database 1303 and a player build process 1305, which may be authoring platform 110.

As an example of system 1300, when response director 210 receives an SMS message from device 130, the response director identifies the device characteristics operator and locale from database 1301 and a Player URL from database 1303 and provides the appropriate Player to the device.

In another embodiment, Player P extend the power of response director 210 by adapting the Application to the resources and limitations of any particular device. Some of these areas of adaptation include the speed of the device's microprocessor, the presence of device resources such as cameras and touch screens. Another area of adaptation is directed to heap, record store and file system memory constraints. In one embodiment, the Player will automatically throttle down an animation to the frame rate that the device can handle so that the best possible user experience is preserved. Other extensions include device specific facilities such as location awareness, advanced touch screen interactions, push extensions, access to advanced phone facilities, and many others

#### Memory Management

In one embodiment, Player P includes a logical page virtual memory manager. This architecture requires no supporting hardware and works efficiently with constrained devices. All page view images, which could span multiple Applications, are placed in a table as highly compacted and compressed code. A typical page view will range from 500 bytes up to about 1,500 bytes. (See, for example, the Rempell patent) When rolled into the heap and instantiated this code increases to the more typical 50,000 up to 250,000 bytes. Additional alert pages may also be rolled into the heap and superimposed on the current page view. Any changes to any page currently downloaded are placed in a highly compact change vector for each page, and rolled out when the page is discarded. Note that whenever an Application is visited that had previously been placed in virtual memory the Server is interrogated to see if a more current version is available, and, if so, downloads it. This means that Application logic can be changed in real-time and the results immediately available to mobile devices.

To operate efficiently with the bandwidth constraints of mobile devices, authoring platform 110 may also utilize anticipatory streaming and multi-level caching. Anticipatory streaming includes multiple asynchronous threads and IO request queues. In this process, the current Application is

36

scanned to determine if there is content that is likely to be required in as-yet untouched page views. Anticipatory streaming also looks for mapping Applications, where the user may zoom or pan next so that map content is retrieved prior to the user requesting it. For mapping applications, anticipatory streaming downloads a map whose size is greater than the map portal size on the device and centered within the portal. Any pan operation will anticipatory stream a section of the map to extend the view in the direction of the pan while, as a lower priority, bring down the next and prior zoom levels for this new geography. Zooming will always anticipatory stream the next zoom level up and down.

Multi-level caching determines the handset's heap through an API, and also looks at the record store to see how much memory is resident. This content is placed in record store and/or the file system, and may, if there is available heap, also place the content there as well. Multi-level caching permits the management of memory such that mobile systems best use limited memory resources. Multi-level caching is a memory management system with results similar to embedding, without the overhead of instantiating the content. In other words, with multi-level caching, handset users get an "embedded" performance without the embedded download. Note that when content is flagged as cacheable and is placed in persistent storage, a digital rights management (DRM) solution will be used.

One embodiment of each of the methods described herein is in the form of a computer program that executes on a processing system. Thus, as will be appreciated by those skilled in the art, embodiments of the present invention may be embodied as a method, an apparatus such as a special purpose apparatus, an apparatus such as a data processing system, or a carrier medium, e.g., a computer program product. The carrier medium carries one or more computer readable code segments for controlling a processing system to implement a method. Accordingly, aspects of the present invention may take the form of a method, an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. Furthermore, the present invention may take the form of carrier medium (e.g., a computer program product on a computer-readable storage medium) carrying computer-readable program code segments embodied in the medium. Any suitable computer readable medium may be used including a magnetic storage device such as a diskette or a hard disk, or an optical storage device such as a CD-ROM.

It will be understood that the steps of methods discussed are performed in one embodiment by an appropriate processor (or processors) of a processing (i.e., computer) system executing instructions (code segments) stored in storage. It will also be understood that the invention is not limited to any particular implementation or programming technique and that the invention may be implemented using any appropriate techniques for implementing the functionality described herein. The invention is not limited to any particular programming language or operating system. It should thus be appreciated that although the coding for programming devices has not been discussed in detail, the invention is not limited to a specific coding method. Furthermore, the invention is not limited to any one type of network architecture and method of encapsulation, and thus may be utilized in conjunction with one or a combination of other network architectures/protocols.

Reference throughout this specification to "one embodiment," "an embodiment," or "certain embodiments" means that a particular feature, structure or characteristic described

US 9,928,044 B2

37

in connection with the embodiment is included in at least one embodiment of the present invention. Thus, appearances of the phrases “in one embodiment,” “in an embodiment,” or “in certain embodiments” in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures or characteristics may be combined in any suitable manner, as would be apparent to one of ordinary skill in the art from this disclosure, in one or more embodiments.

Throughout this specification, the term “comprising” shall be synonymous with “including,” “containing,” or “characterized by,” is inclusive or open-ended and does not exclude additional, unrecited elements or method steps. “Comprising” is a term of art which means that the named elements are essential, but other elements may be added and still form a construct within the scope of the statement. “Comprising” leaves open for the inclusion of unspecified ingredients even in major amounts.

Similarly, it should be appreciated that in the above description of exemplary embodiments, various features of the invention are sometimes grouped together in a single embodiment, figure, or description thereof for the purpose of streamlining the disclosure and aiding in the understanding of one or more of the various inventive aspects. This method of disclosure, however, is not to be interpreted as reflecting an intention that the claimed invention requires more features than are expressly recited in each claim. Rather, as the following claims reflect, inventive aspects lie in less than all features of a single foregoing disclosed embodiment, and the invention may include any of the different combinations embodied herein. Thus, the following claims are hereby expressly incorporated into this Mode(s) for Carrying Out the Invention, with each claim standing on its own as a separate embodiment of this invention.

Thus, while there has been described what is believed to be the preferred embodiments of the invention, those skilled in the art will recognize that other and further modifications may be made thereto without departing from the spirit of the invention, and it is intended to claim all such changes and modifications as fall within the scope of the invention. For example, any formulas given above are merely representative of procedures that may be used. Functionality may be added or deleted from the block diagrams and operations may be interchanged among functional blocks. Steps may be added or deleted to methods described within the scope of the present invention.

We claim:

1. A system for generating code to provide content on a display of a device, said system comprising:  
computer memory storing:

- a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, where each symbolic name has an associated data format class type corresponding to a subclass of User Interface (UI) objects that support the data format type of the symbolic name, and where each symbolic name has a preferred UI object, and
- b) an address of the web service;

an authoring tool configured to:

define a UI object for presentation on the display,

where said defined UI object corresponds to a web component included in said computer memory selected from a group consisting of an input of the

38

web service and an output of the web service, where each defined UI object is either:

- 1) selected by a user of the authoring tool; or
- 2) automatically selected by the system as the preferred UI object corresponding to the symbolic name of the web component selected by the user of the authoring tool,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object, where the selected symbolic name is only available to UI objects that support the defined data format associated with that symbolic name,

store information representative of said defined UI object and related settings in a database;

retrieve said information representative of said one or more said UI object settings stored in said database; and

build an application consisting of one or more web page views from at least a portion of said database utilizing at least one player, where said player utilizes information stored in said database to generate for the display of at least a portion of said one or more web pages,

wherein when the application and player are provided to the device and executed on the device, and

when the user of the device provides one or more input values associated with an input symbolic name to an input of the defined UI object, the device provides the user provided one or more input values and corresponding input symbolic name to the web service, the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,

and the player receives the output symbolic name and corresponding one or more output values and provides instructions for the display of the device to present an output value in the defined UI object.

2. The system of claim 1, where said system stores information in a registry, and wherein the registry includes definitions of input and output related to said web service.

3. The system of claim 1, where said web component is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

4. The system of claim 1, where said UI object is an input field for a chat.

5. The system of claim 1, where said UI object is an input field for a web service.

6. The system of claim 1, where said UI object is an input field usable to obtain said web component, where said input field includes a text field, a scrolling text box, a check box, a drop down-menu, a list menu, or a submit button.

7. The system of claim 1, where said web component is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

8. The system of claim 1, where said authoring tool is further configured to:

define a phone field or list; and

generate code that, when executed on the device, allows a user to supply a phone number to said phone field or list.

9. The system of claim 1, where said authoring tool is further configured to:

US 9,928,044 B2

39

define a SMS field or list; and  
generate code that, when executed on the device, allows  
a user to supply an SMS address to said SMS field or  
list.

10. The system of claim 1,  
where said code includes three or more codes, where one  
of said three or more codes is device specific, and  
where two of said three or more codes is device  
independent.

11. The system of claim 1, where said code is provided  
over said network.

12. The system of claim 1, wherein said defined UI object  
corresponds to a widget.

13. The system of claim 1, where said player is activated  
and runs in a web browser.

14. The system of claim 1, where said player is a native  
program.

15. A method of displaying content on a display of a  
device having a player and non-volatile computer memory  
storing symbolic names required for evoking one or more  
web components each related to a set of inputs and outputs  
of a web service obtainable over a network, where the  
symbolic names are character strings that do not contain  
either a persistent address or pointer to an output value  
accessible to the web service, where each symbolic name  
has an associated data format class type corresponding to a  
subclass of User Interface (UI) objects that support the data  
format type of the symbolic name, and where each symbolic  
name has a preferred UI object, and an address of the web  
service, said method comprising:

defining a UI object for presentation on the display, where  
said UI object corresponds to a web component  
included in the computer memory, where said web  
component is selected from a group consisting of an  
input of a web service and an output of the web service,  
where each defined UI object is either: 1) selected by a  
user of the authoring tool; or 2) automatically selected  
by the system as the preferred UI object corresponding  
to a symbolic name of the web component selected by  
the user of the authoring tool;

selecting the symbolic name corresponding to the web  
component of the defined UI object;

associating the selected symbolic name with the defined  
UI object, where the selected symbolic name is only  
available to UI objects that support the defined data  
format associated with that symbolic name;

storing information representative of said defined UI  
object and related settings in a database;

retrieving said information representative of said one or  
more said UI object settings stored in said database;  
and

building an application consisting of one or more web  
page views from at least a portion of said database  
utilizing the player, where said player utilizes information  
stored in said database to generate for the display  
of at least a portion of said one or more web pages,

40

wherein, when the application and player are provided to  
the device and executed on the device, and when the  
user of the device provides one or more input values  
associated with an input symbolic name to an input of  
the defined UI object, 1) the device provides the user  
provided one or more input values and corresponding  
input symbolic name to the web service, 2) the web  
service utilizes the input symbolic name and the user  
provided one or more input values for generating one or  
more output values having an associated output sym-  
bolic name, and 3) the player receives the output  
symbolic name and corresponding one or more output  
values and provides instructions for the display of the  
device to present an output value in the defined UI  
object.

16. The method of claim 15, where said method stores  
information in a registry, and wherein the registry includes  
definitions of input and/or output related to said web service.

17. The method of claim 15, where said web component  
is a text chat, a video chat, an image, a slideshow, a video,  
or an RSS feed.

18. The method of claim 15, where said UI object is an  
input field for a chat.

19. The method of claim 15, where said UI object is an  
input field for a web service.

20. The method of claim 15, where said UI object is an  
input field usable to obtain said web component, where said  
input field includes a text field, a scrolling text box, a check  
box, a drop down-menu, a list menu, or a submit button.

21. The method of claim 15, where said web component  
is an output of a web service, is the text provided by one or  
more simultaneous chat sessions, is the video of a video chat  
session, is a video, an image, a slideshow, an RSS display,  
or an advertisement.

22. The method of claim 15, further comprising:

defining a phone field or list; and  
generating code that, when executed on the device, allows  
a user to supply a phone number to said phone field or  
list.

23. The method of claim 15, further comprising:

defining a SMS field or list; and  
generating code that, when executed on the device, allows  
a user to supply an SMS address to said SMS field or  
list.

24. The method of claim 15, and such that said player  
interprets dynamically received, device independent values  
of the web component defined in the application.

25. The method of claim 15, further comprising:

providing said application and player over said network.

26. The method of claim 15, wherein said UI object  
corresponds to a widget.

27. The method of claim 15, where said player is activated  
and runs in a web browser.

28. The method of claim 15, where said player is a native  
program.

\* \* \* \* \*

## CERTIFICATE OF COMPLIANCE

I hereby certify that this brief complies with the type-volume limitations of Fed. Cir. R. 32(a). This brief contains 13,731 words (including diagrams and images), excluding the parts of the brief exempted by Fed. R. App. P. 32(f) and Fed. Cir. R. 32(b), as counted by Microsoft® Word 2010, the word processing software used to prepare this brief.

This brief complies with the typeface requirements of Fed. R. App. P. 32(a)(5) and the type style requirements of Fed. R. App. P. 32(a)(6). This brief has been prepared in a proportionally spaced typeface using Microsoft® Word 2010, Times New Roman, 14 point.

Dated: November 13, 2023

/s/ James R. Nuttall

James R. Nuttall  
(Lead Counsel)  
jnuttall@steptoe.com  
Katherine H. Tellez  
Robert F. Kappers  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300  
Facsimile: (312) 577-1370

*Counsel for Plaintiff-Appellant,  
Express Mobile, Inc.*

### **CERTIFICATE OF SERVICE**

I hereby certify that, on the 13th day of November, 2023, I electronically filed the foregoing with the Clerk of Court using the CM/ECF system which thereby served a copy upon all counsel of record.

Upon acceptance by the Court of the e-filed document, six paper copies of the brief will be filed with the Court via Federal Express, priority overnight, within the time provided in the Court's rules.

November 13, 2023

/s/ James R. Nuttall

James R. Nuttall  
(Lead Counsel)  
jnuttall@steptoe.com  
Katherine H. Tellez  
Robert F. Kappers  
STEPTOE & JOHNSON LLP  
227 West Monroe Street, Suite 4700  
Chicago, IL 60606  
Telephone: (312) 577-1300  
Facsimile: (312) 577-1370

*Counsel for Plaintiff-Appellant,  
Express Mobile, Inc.*